

EXPERTISE-HALL SERIES IN PERSONAL COMPUTING

JOHN E. UFFENBECK

Hardware Interfacing with the Apple II Plus



*A step by step introduction to microcomputer interfacing—
featuring 13 practical hardware experiments.*



**HARDWARE
INTERFACING
WITH THE
APPLE II PLUS**

HARDWARE INTERFACING WITH THE APPLE II PLUS

JOHN E. UFFENBECK

PRENTICE-HALL, INC., *Englewood Cliffs, NJ* 07632

Library of Congress Cataloging in Publication Data

Uffenbeck, John E.

Hardware interfacing with the Apple II Plus.

Includes index.

1. Computer interfaces—Experiments. 2. Apple II (Computer)—Experiments. I. Title.

TK7887.5.U326 1983 621.3819'583 83-3186

ISBN 0-13-383851-X

ISBN 0-13-383844-7 (pbk.)

Apple is a registered trademark of Apple Computer, Inc.

Editorial/production supervision: Kathryn Gollin Marshak

Cover design: Diane Saxe

Manufacturing buyer: Gordon Osbourne

Cover photograph courtesy of Apple Computer, Inc.

© 1983 by Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2

0-13-383844-7 {P}

0-13-383851-X {C}

Prentice-Hall International, Inc., *London*

Prentice-Hall of Australia Pty. Limited, *Sydney*

Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Prentice-Hall Canada Inc., *Toronto*

Prentice-Hall of India Private Limited, *New Delhi*

Prentice-Hall of Japan, Inc., *Tokyo*

Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*

Whitehall Books Limited, *Wellington, New Zealand*

For Kathy

CONTENTS

PREFACE

x

PART 1 GETTING STARTED

1

Introduction	3
Hardware Requirements	3
Nice-to-Have Components	7
Software Requirements	9
Conclusion	9

PART 2 BASIC CONCEPTS OF MICROCOMPUTER INPUT/OUTPUT

11

Experiment 1	The PEEK and POKE Commands	13
Experiment 2	Address Decoding	22
Experiment 3	Memory-Mapped Output Port Concepts	36
Experiment 4	Memory-Mapped Input Port Concepts	46
Experiment 5	Interfacing the 8255 Programmable Peripheral Interface	56
Experiment 6	Interfacing the 6820 Programmable Interface Adapter	69

PART 3
SPECIAL INTERFACING PROBLEMS

83

Experiment 7	Hardware Interfacing Techniques, Part 1: Inputs	85
Experiment 8	Hardware Interfacing Techniques, Part 2: Outputs	100
Experiment 9	Interfacing a Digital-to-Analog Converter	114
Experiment 10	Interfacing an Analog-to-Digital Converter	126
Experiment 11	Handshaking I/O	140
Experiment 12	Serial Interfacing	153
Experiment 13	Interfacing a Programmable Sound Generator	172

APPENDICES

191

A	Wiring the Vector Card	191
B	Parts List	195
C	Binary and Decimal Numbers	199
D	Basic Logic Gates	202
E	JK Flip-Flop	205
F	If the Experiment Doesn't Work	207
G	Data Sheets	210

INDEX

235

PREFACE

This book is written for the computer “*hacker*,” the technician, the engineer, and in general anyone with an interest in knowing a bit more about the electronics of the Apple II Plus personal computer system. In particular it deals with interfacing the sometimes “hostile” outside world to the expansion bus of the Apple.

If you have never thought of using the Apple to control your world, consider some of the possibilities. Using simple BASIC software you can monitor the indoor or outdoor temperature, the opening and closing of a door, the level of light in a room, the status of a smoke detector, or the voltage on a charging capacitor. Once input, you can process this information and turn on the light across the room, switch the furnace off or on, generate a piercing alarm signal, or program the “tinkling” of a bell.

All of these interfacing projects and several more are described in the 13 experiments in **HARDWARE INTERFACING WITH THE APPLE II PLUS**. You will learn interfacing by doing interfacing. The book begins with an explanation of the **PEEK** and **POKE** commands as their use will be critical throughout. From there you will learn *address decoding* and input and output ports. The Intel 8255 programmable peripheral interface is used in Experiment 5 and contrasted with the Motorola MC6820 peripheral interface adapter in Experiment 6. These two ICs greatly simplify the job of interfacing to the outside world.

The last seven experiments deal with special interfacing techniques such as analog-to-digital and digital-to-analog conversion methods. Also covered is serial interfacing and the concept of “handshaking” logic. The book ends with a programmable sound generator project requiring 8 ICs. This circuit is capable of producing a fantastic array of sounds from train whistles to gun shots to an electronic organ.

You will be assumed to be a proficient—but not expert—BASIC programmer. Although machine language could be used for any of the experiments, BASIC is used throughout. You should be familiar with digital bread-

boarding techniques and the 7400 series of digital logic gates. *Mixed logic* symbology is used for all logic elements, and this technique is reviewed in Appendix D.

The experiments are organized so that you can quickly see what you will be required to do—the OVERVIEW, what you will learn; the OBJECTIVES, what parts you will need; the PARTS LIST; a DISCUSSION on the theory; and the experimental PROCEDURE itself.

One of the most difficult parts of any electronics project today is locating the components and getting started. Part 1 addresses this issue and Appendix B provides a complete parts list for all of the experiments with several sources.

If you have never used your Apple as a hardware controller before, be prepared for a pleasant surprise. You may find hardware interfacing even more challenging and exciting than that last BASIC program you wrote.

JOHN E. UFFENBECK

ACKNOWLEDGMENTS

The following figures and tables from *Hardware Interfacing with the TRS-80* by John Uffenbeck are reproduced with the permission of Prentice-Hall, Inc. Figure numbers refer to *Hardware Interfacing with the Apple II Plus*.

Figure I-4, Tables I-1, I-2, Figures 1-1, 2-4, 3-3, Table 3-1, Figures 4-1, 4-3, 4-4, 5-3, 5-6, 7-1, 7-2, 7-4, 7-5, 7-6, 7-8, 7-9, 8-1, 8-2, 8-4, 8-5, 8-6, 8-7, 9-1, 9-4, 9-5, 10-1, 10-2, 10-3, 10-4, 10-5, 11-1, 11-2, 11-4, 11-5, 11-7, 12-1, 12-2, 12-3, 12-5, 12-9, Table 12-2, Figures 13-1, 13-2, 13-5, 13-7, 13-9, 13-11, Tables 13-1, C-1, Figures D-1, D-2, D-3, D-4, E-1, E-2, F-1.

The data sheets in Appendix G are reproduced courtesy of Intel Corporation, Motorola, Inc., General Instrument Corporation, and National Semiconductor Corporation.

HARDWARE INTERFACING WITH THE APPLE II PLUS

1

Getting Started

Read this section before attempting any of the experiments in this book. You will need special hardware to do the experiments and this section tells you what to get and where you can find it.

INTRODUCTION

The Apple II plus microcomputer shown in Fig. I-1 is one of the most popular microcomputers on the market today. Because of this there are hundreds of computer programs and operating systems available for the Apple. These range from *Star Invaders* to *Visicalc* to *Pascal*. A nearly equal number of books have been written describing how to use and operate these programs using the Apple's built-in BASIC language, *Applesoft*.

This book, however, deals with that area of microcomputers that is sometimes considered "black magic" or "taboo" by the microcomputer owner: namely, microcomputer hardware and interfacing. It is my hope to clear the air by leading you through 13 progressively more complex hardware experiments on your Apple personal computer. By the time you have completed all 13 experiments, you may not be an expert, but you will certainly understand microcomputer input and output. And you will also appreciate the potential that your computer has as a hardware controller in the home or industry.

Before beginning these experiments you should acquaint yourself with the basic hardware and software necessary to perform the experiments. These are explained in the following sections.

HARDWARE REQUIREMENTS

All of the experiments in this book are written with the Apple II computer in mind and all hardware details are specific to this computer. The Apple is most commonly available in the Apple II Plus version, with 48K bytes of RAM (random access read-write memory) and one disk drive. While this system is more than sufficient, the minimum configuration is assumed to be:

1. Any Apple II computer with Applesoft.
2. At least 8K of RAM.
3. A video monitor or suitable television receiver.



Figure I-1 Apple II Plus microcomputer system. All of the experiments in this book are intended for this computer. Reprinted from the Apple II Reference Manual with the permission of Apple Computer, Inc.

Notice that no disk drive, printer, language card, I/O card, or other peripheral is required. If you have these things, fine, but if not, you won't need them.

What you will need is some means of connecting to one of the peripheral connector slots (preferably slot 7) and a *solderless breadboard* for building the interface circuits. The next few subsections detail these hardware requirements.

THE RIBBON CABLE ASSEMBLY AND INTERFACE CARD

Figure I-2a illustrates the basic interfacing concept to be used throughout this book. A Vector Electronics 4609 *plugboard* is inserted into slot 7 of the Apple's motherboard. The 40-pin connector on the rear of this card is then wired to access the important interfacing signals provided by the Apple at the slot 7 connector. This is shown in Fig. I-2b. A length of 40-conductor ribbon cable with a card edge connector at one end (the computer end) and a socket connector on the other is used to bring these signals out to the motherboard. The socket connector conveniently allows No. 22 or 24 gauge wire connections between it and the breadboard.

The Vector card is available from Jameco Electronics (1355 Shoreway Road, Belmont, CA 94002), Priority One Electronics (9161 Deering Avenue, Chatsworth, CA 91311), and several other suppliers for about \$25. The 40-



Figure I-2 (a) Apple computer with its cover removed, revealing the Vector Electronics plugboard in slot 7. The ribbon cable connected to the rear of this card provides access to the interfacing signals required to do the experiments. A solderless breadboard is used to build up the various interfacing circuits.

conductor ribbon cable is available from Digi-Key Corp. (P.O. Box 677, Thief River Falls, MN 56701). The part number is 924150-24 and the approximate cost is \$15.

Before proceeding to Experiment 1 you should wire the Vector card following the instructions given in Appendix A. This appendix also gives a complete description of the electrical signals available at each peripheral slot connector and at the breadboard.

One particularly nice feature of the Apple computer is that ± 5 V and ± 12 V are available at each peripheral connector. This means that it will not

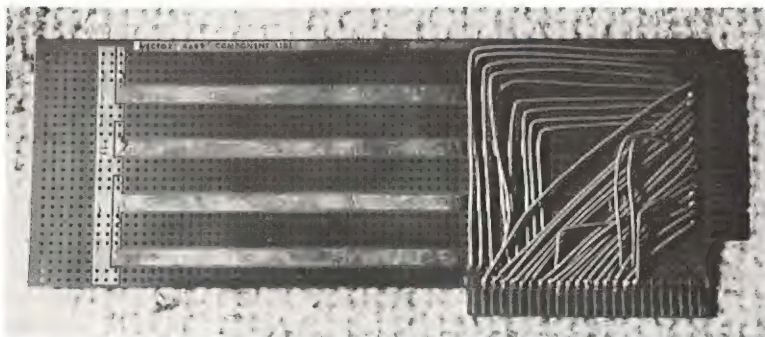


Figure I-2 (b) Jumper wires are used to bring the interfacing signals from the Apple's peripheral connector to the edge connector on the Vector card.

be necessary for you to use an external power supply to perform any of the experiments. The current capacities of these voltages are limited, however, and it may be wise to remove any extra cards you have in your computer when using the Vector card. But you should be able to operate with three or four other cards in the computer besides the interface board (see Appendix F, "If the Experiment Doesn't Work").

SOLDERLESS BREADBOARD

The purpose of the breadboard is to allow easy (solderless) connections between the various integrated circuits (ICs) used in the experiments and the Apple. Referring to Fig. I-3, the ICs are placed so as to straddle the center divider of the socket. There are then four remaining connections to each IC lead in each row. Connections on one side of the center notch are electrically isolated from the other side.

Table I-1 compares several types of solderless breadboards. The more rows per side, the more ICs you will be able to interconnect and the more expensive the breadboard. The first two entries in the table provide sockets only, whereas the last three have the sockets mounted on a small base (see Fig. I-3). If the cost of some of the Proto-Boards seems high, you may want to consider fabricating your own base. In this way you can expand your breadboard with additional sockets as needed while holding down the initial cost.

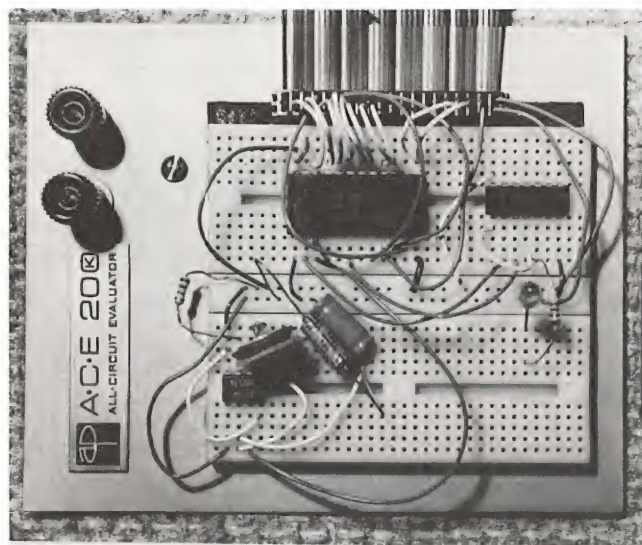


Figure I-3 Breadboard end of the ribbon cable assembly. The two sockets shown provide 68 rows of connections with a power bus (+5 V and ground) running between the two. The ribbon cable socket connector is held in place by the "stickum" on the base plate. The cable itself is a 40-conductor ribbon with a card edge connector on the computer end. Refer to the text for sources for these components.

TABLE I-1 SOLDERLESS BREADBOARDS ARE AVAILABLE IN A VARIETY OF CONFIGURATIONS AND PRICES

Type	Number of rows per side	Part number	Manufacturer
Modular breadboard socket (may be expanded)	47	276-174	Radio Shack about \$10
Quick Test Socket and bus strip (may be expanded)	59	QT-59S, QT-59B	Jameco Electronics 1355 Shoreway Road Belmont, CA 94002 about \$15
Proto-Board with Quick Test Sockets	68	ACE-200K, 923333	Digi-Key Corp. P.O. Box 677 Thief River Falls, MN 56701 about \$20
Proto-Board with Quick Test Sockets	96	ACE-201K, 923334	Digi-Key about \$30
Proto-Board with Quick Test Sockets	94	PB-102	Jameco about \$35

COMPONENTS

In general, each experiment will require you to wire one or more integrated circuits on your breadboard. These circuits are then connected to the ribbon cable's socket connector with No. 22 or 24 gauge solid insulated wire. Figure I-3 illustrates the method. Note that in addition to the ICs, several other components may be required, depending on the experiment. Appendix B lists the components needed to do all the experiments in this book. Each project also begins with a complete list of parts needed for that experiment.

NICE-TO-HAVE COMPONENTS

There are additional supplies that can make your journey through this book more meaningful and be an aid to you in troubleshooting your circuits.

LOGIC PROBE

A digital circuit may have only two operating states: ON or OFF. The output of a logic gate is either high (1) or low (0). Although we could monitor this condition with a voltmeter, another tool is often used—a *logic probe*. Figure I-4 pictures a typical probe. The light-emitting diodes (LEDs) serve as the logic-level indicators. These correspond to the logic 1 and 0 states and



Figure I-4 A logic probe will be handy for monitoring logic levels. Its three LEDs brightly indicate a logic 1, a logic 0, or a pulse condition.

to a pulse condition (a logic level rapidly switching between the 1 and 0 states).

A logic probe will be very handy for monitoring circuit conditions in the various experiments in this book. Often we will be programming the Apple to output a certain binary pattern which can easily be monitored with the logic probe. Table I-2 lists several common logic probes, their distinguishing features, and suppliers.

Again you can save money by building your own. There have been numerous articles on logic probe construction over the last few years: for example, Steve Dominguez, "Probos V," *Kilobaud Microcomputing*, October 1979, p. 78. In this article Dominguez describes a logic probe with capabilities equal to those in Table I-2 but costing less than \$5 for parts!

TABLE I-2 VARIOUS COMMONLY AVAILABLE LOGIC PROBES^a

Source	Part number	Description
Radio Shack about \$25	22-301	Fully assembled, detects pulses to 50 ns
Jameco Electronics 1355 Shoreway Road Belmont, CA 94002 about \$23	LPK-1	Logic probe kit, detects pulses to 300 ns; $f_{\max} = 1.5$ MHz
Jameco about \$50	LP-1	Fully assembled, detects pulses to 50 ns; $f_{\max} = 10$ MHz

^aA logic probe, although not essential, will be handy for monitoring logic levels in the interface circuits.

Another interesting example is the logic probe described by Robert Kreiger, "Build an Audible Logic Probe," *Popular Electronics*, July 1980, p. 73. This circuit produces different audio tones for the three logic conditions (high, low, and pulse).

DC VOLTMETER

Although not essential, a DC voltmeter will be handy to measure power supply voltages and to troubleshoot simple electrical circuits. It is particularly handy to have an ohms function for continuity testing in cables-testing for open circuits and short circuits. A meter with volts, ohms, and current ranges is generally called a *VOM*. There are numerous sources for such meters, with prices ranging from \$5 to several hundred dollars.

OSCILLOSCOPE

A dual-channel oscilloscope would be very useful for troubleshooting and tracing pulses through the various experiments. With it you would be better able to appreciate the dynamic nature of the interfacing circuits. If you have one, great—be sure to use it. If not, don't worry, all the experiments can easily be built and tested without one.

SOFTWARE REQUIREMENTS

All the programs in this book are written in BASIC. Machine language could be used for the software, but BASIC is much easier to work with and usually does the job just as well. Occasionally, a problem can be solved only by using machine code. Although these areas are pointed out, they are left for another book and another day.

CONCLUSION

As a starting point, I would recommend purchasing a 25-ft roll of No. 24 gauge solid insulated wire and a good-quality wire stripper. Then page through the experiments in Parts 2 and 3 and decide if you want to do all or just some of these experiments. Use Appendix B to make up a parts shopping list. Don't forget about the ribbon cable and Vector interface card. The addresses of several suppliers are also given in Appendix B. Now prepare to enter the world of microcomputer interfacing.

2

Basic Concepts of Microcomputer Input/Output

The six experiments in this section will teach you how to add input and output ports to your Apple. BASIC software for controlling these ports is also discussed.

EXPERIMENT 1

THE PEEK AND POKE COMMANDS

OVERVIEW

In this experiment you will connect the ribbon cable assembly to the breadboard. A 7476 JK flip-flop will then be wired to monitor memory read and memory write cycles.

OBJECTIVES

The key points to be learned from this experiment are:

1. For interfacing purposes, the Apple can be thought of as having three buses: the data bus, the address bus, and the control bus.
2. The control bus consists of two signals or lines: R/\overline{W} and $\Phi 0$.
3. The Apple treats all its I/O devices as if they were memory locations. This is called *memory-mapped I/O*.
4. Only two commands in BASIC allow access to the I/O devices. These are the **PEEK** and **POKE** commands.

PARTS LIST

- 1 7476 JK flip-flop
- 1 7400 quad two-input NAND gate
- 1 LED (light-emitting diode)
- 1 180- Ω resistor (brown-gray-brown)
- 1 10- μ F capacitor

DISCUSSION: THE THREE-BUS SYSTEM ARCHITECTURE

Throughout this book we will be working with digital signals output by various integrated circuits (ICs) in our interface circuits. Learning to understand these signals and their dynamic nature is the essence of microcomputer interfacing.

But what is a *digital signal*? Is it something that can be observed only with an expensive oscilloscope, or will we need a digital voltmeter? First you must appreciate that a digital signal can take on only one of two values. These are usually referred to as the 1 and 0 states. Other names are HIGH and LOW or ON and OFF.

Most of the ICs we will be working with are from the TTL (transistor-transistor logic) family of logic circuits. These devices typically produce 3.4 V for a logic 1 and 0.2 V for a logic 0. Later in this experiment we will see how light-emitting diodes (LEDs) can be used to observe these digital signals.

Because a single digital line cannot carry too much information, several lines are usually used and collectively referred to as a *bus*. On the Apple we have 16 lines devoted to carrying memory addresses and refer to these lines as the *address bus*. We also have eight lines used for carrying data to and from the microprocessor. These lines are referred to as the *data bus*. We will see that a third bus is also necessary for controlling the direction of data flow and basic timing. This bus is referred to as the *control bus*.

Interfacing to any microcomputer requires a detailed understanding of these three buses and their unique timing. This is the primary objective of Experiments 1 through 4.

The Apple System Block Diagram

Figure 1-1 illustrates the Apple computer in block diagram form (actually it could represent nearly any 8-bit microcomputer system). The main components of the system are illustrated. The clock generator establishes the basic timing for the entire computer. The 6502 CPU (central processing unit) acts as a "traffic cop" directing the flow of data between the various system components as it is directed by programs stored in RAM (random access memory) and ROM (read-only memory). The later two blocks can be thought of as programmable user memory and permanent operating system memory, respectively. The keyboard and video screen are actually memory-mapped devices and can be considered part of the RAM block.

Each of the three buses shown has a specific responsibility with which we must become thoroughly familiar.

1. The *data bus*. These eight lines handle the actual 8 bits of information (one byte) that the CPU is working with. These bits may represent data

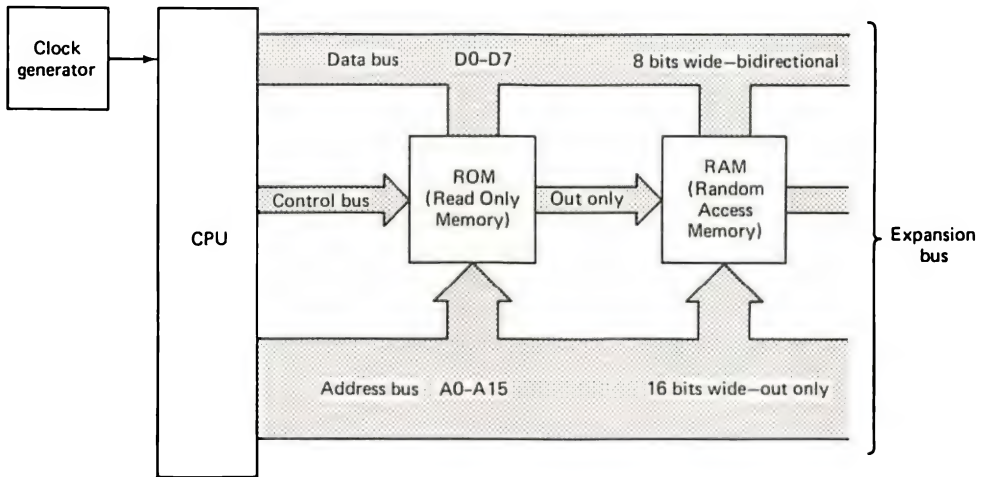


Figure 1-1 Apple computer system block diagram. All components in the system interface to the 6502 CPU over the 8-bit data bus. Refer to Appendix A and the text for specific details on signals brought out to the peripheral connectors.

bytes or instruction operation codes.

2. The *address bus*. For the 6502 (and most other microprocessors) the address bus consists of 16 output lines. The binary pattern placed on these lines by the CPU determines the specific memory location to be read from or written to. Because there are 16 address lines, there are 2^{16} or 65,536 different addresses possible. These range from all 16 lines low (address 0) to all 16 lines high (address 65535). The specific pattern on these lines is always generated by the CPU (except for direct memory access) and therefore these lines are outputs only.
3. The *control bus*. This bus could be considered to be all remaining signals at a typical peripheral connector. But this would mean a 21-pin bus (excluding the five power pins)! For our immediate purposes we will need only two of these lines (we will add a third in the next experiment). The first is R/\overline{W} . This control signal indicates if the current operation (machine cycle) will require the CPU to read data (an input) or write data (an output). The solid bar above the W implies that the write operation occurs when this line is low, a read when it is high. $\Phi 0$ is the second control signal and it is the microprocessor clock signal. In some literature $\Phi 0$ is designated $\Phi 2$. This signal is important because the 6502 microprocessor uses its data and address buses only when this line is high.

The three-bus structure just described requires a total of 27 lines. There are also five power pins (± 5 V, ± 12 V, and ground). The remaining 18 pins

are described in Appendix A but will not be required for any of the experiments in this book.

Read and Write Operations

The clock generator in Fig. 1-1 uses a 14.318-MHz crystal for its time base. Flip-flop circuits divide this down to 1.023 MHz and it is then applied to the 6502 microprocessor as clock signal $\Phi 0$. Internally, the $\Phi 0$ clock signal is split into two new clocks, called $\Phi 1$ and $\Phi 2$. These two clock phases are then output by the microprocessor and become part of the system control bus. The timing relationships among these three clock signals are shown in Fig. 1-2.

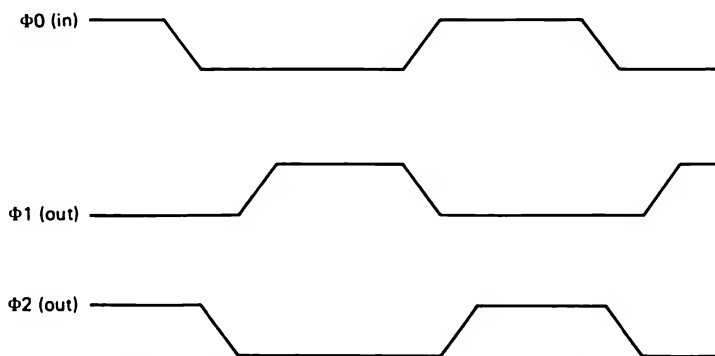


Figure 1-2 Timing relationships between the three clock signals in the Apple computer. Only $\Phi 0$ and $\Phi 1$ are available at the peripheral connectors.

For reasons known only to Apple Computer Corp., the $\Phi 2$ clock output of the 6502 is not used. Only $\Phi 0$ and $\Phi 1$ are available at the peripheral connectors. Because $\Phi 0$ is nearly identical to $\Phi 2$ ($\Phi 0$ leads $\Phi 2$ by approximately 30 ns), there are normally no problems in interfacing to the Apple control bus.

Figure 1-3 illustrates the system timing for both read and write machine cycle types. In Fig. 1-3a a read operation is occurring. The CPU begins by establishing the proper binary address of the memory cell or I/O device on address lines A0 through A15. At the same time, the R/\overline{W} line is brought high to signify a read operation. When R/\overline{W} is high AND $\Phi 0$ is high, valid data is placed on the data bus by the I/O or memory device.

In Fig. 1-3b a write operation is occurring. Again the CPU begins by outputting the memory or I/O device address on A0 through A15. This time R/\overline{W} is pulsed low, signifying a write operation. When R/\overline{W} is low AND $\Phi 0$ high, the CPU places the actual output data on the data bus, allowing memory or I/O to latch the data byte.

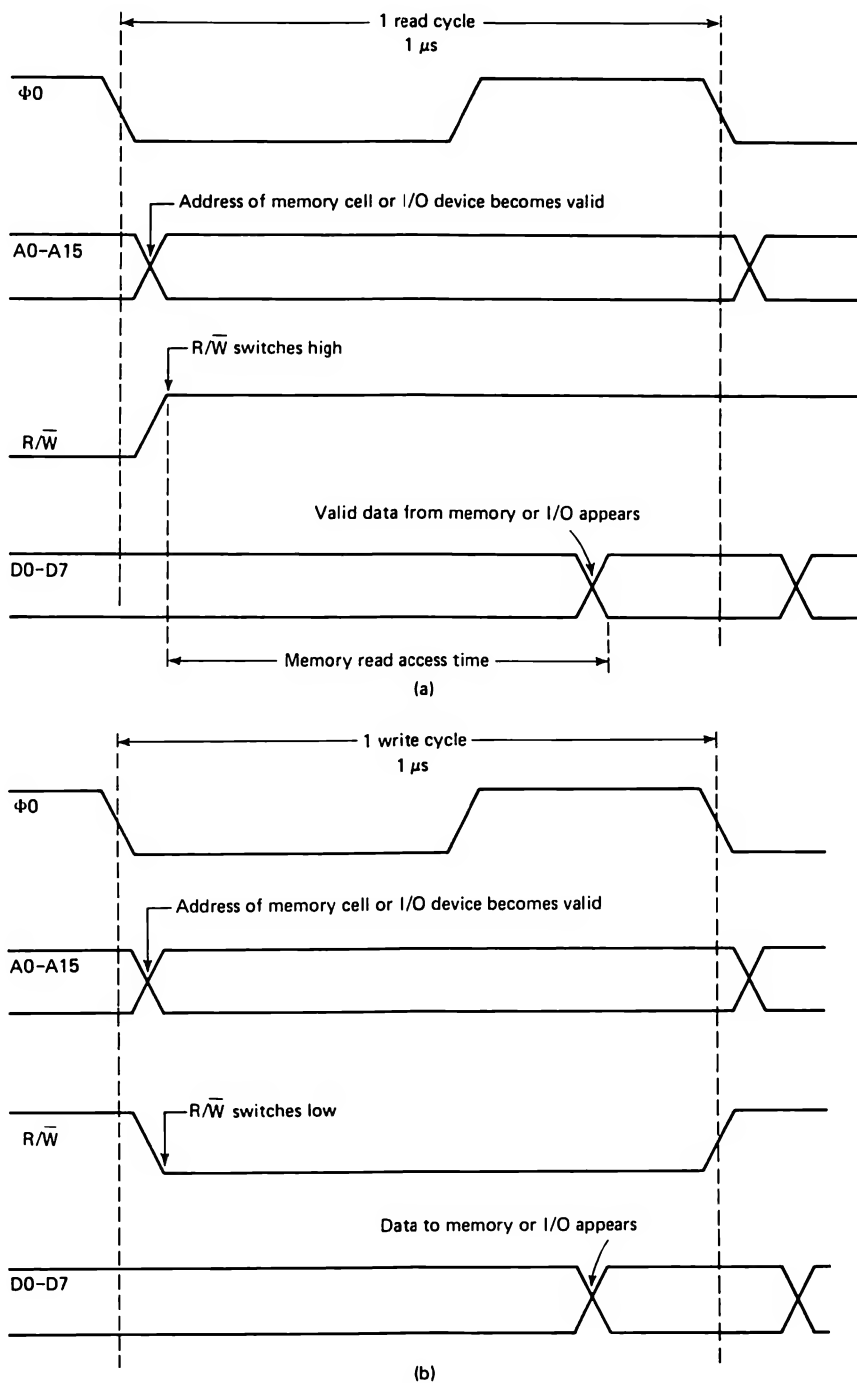


Figure 1-3 Illustrating memory read (a) and memory write (b) machine cycles. In both cases the transfer of data occurs over the data bus and is synchronized with the logic 1 level of $\Phi 0$.

Types of I/O

In general, a microprocessor may interface with its I/O devices in two ways. In the first, called *I/O-mapped I/O*, the microprocessor must use special input (INP) and output (OUT) commands. For example, data is input with a BASIC command such as

```
10 Y=INP(47)
```

or output with a command such as

```
10 OUT 47,Y
```

In the first case the variable Y receives the data from port 47, whereas in the second case the variable Y is output to port 47.

Unfortunately, the 6502 microprocessor (and therefore the Apple computer) does not support this type of I/O. This is a design feature of the microprocessor itself and cannot be changed. For this reason Applesoft does not allow the INP and OUT commands.

The second type of I/O is called *memory-mapped I/O*. In this scheme I/O devices are interfaced so as to appear to be memory locations and the full 16-bit address bus is used. In this case, any command that reads or writes to memory becomes an I/O instruction. In BASIC there are just two. For example, to read the contents of port 26128, we would use

```
10 Y=PEEK(26128)
```

Similarly, to write the contents of Y to this same port we would use

```
10 POKE 26128,Y
```

The experiments to follow will give more detail on how to use these PEEK and POKE commands with memory-mapped I/O devices.

PROCEDURE

Step 1. Refer to Appendix A and wire your Vector card according to the instructions given there. Now insert the card into the Apple slot 7. The cable should exit to the left when viewed from the front of the Apple. See Fig. I-2 for the proper orientation.

Step 2. Because all the experiments require connections to the socket end of the ribbon cable, it will be best to secure this end of the cable to your breadboard. Refer to Fig. I-3. The breadboard sockets have been offset just slightly, allowing $\frac{1}{4}$ inch of the "stickum" on the base to be exposed. This, in

turn, is used to hold the socket end of the cable in place. Figure 1-4b illustrates the proper orientation and the resulting pin number configuration for the socket connector.

Step 3. Connect a short wire from pin 1 of the socket connector to one of the bus strips on the breadboard (label this strip +5 V). Connect a second piece of wire from pin 40 to a second bus strip (label this ground). Now connect a 10- μ F capacitor (+ lead to +5 V) between the two bus strips.

Note. Your breadboard is now in its permanent configuration and ready to perform all the experiments in this book.

Step 4. Refer to Fig. 1-4a and wire this circuit on your breadboard. Figure 1-4b illustrates the proper method for inserting the ICs. Please be sure to observe the following:

1. Position the two ICs so that they straddle the center notch of the breadboard (see Figs. 1-3 and 1-4b).
2. The ICs must have power; +5 V on pin 14 (7400) and pin 5 (7476). Be sure also to connect the grounds: pin 7 (7400) and pin 13 (7476).
3. Connect the LED with the lead from the flat side of the package to pin 15 of the 7476. Connect a 180- Ω resistor (brown-gray-brown) from the other LED lead to the +5 V bus strip.
4. Connect a wire from pin 37 ($\Phi 0$) of the socket connector to pin 1 of the 7400. Connect pin 38 (R/\bar{W}) to pin 2 of the 7400.

Note. When $\Phi 0$ AND R/\bar{W} are both high (a valid memory read cycle) the 7400 output will go low. This will act as a clock pulse to the 7476, which is wired in the toggle mode (refer to Appendix E for a review of the JK flip-flop). Each time a *memory read* occurs, the 7476 will switch states and the LED will toggle ON and OFF.

Step 5. Type and run the following program and observe the LED.

```
10 Y=PEEK(26128)
20 GOTO 10
```

Question 1-1. The LED should appear to be brightly lit. Can you explain why? (*Hint:* Remove and replace the wire to the 7476 clock input (pin 1) several times. You should see that the LED brilliance dims slightly when the connection is made. Also, when the wire is removed the LED may be ON or OFF, proving that the output is switching. If you have a logic probe or oscilloscope you can monitor the flip-flop input and output pins and observe this pulse condition.*)

*Answers to all questions are provided at the end of each experiment.

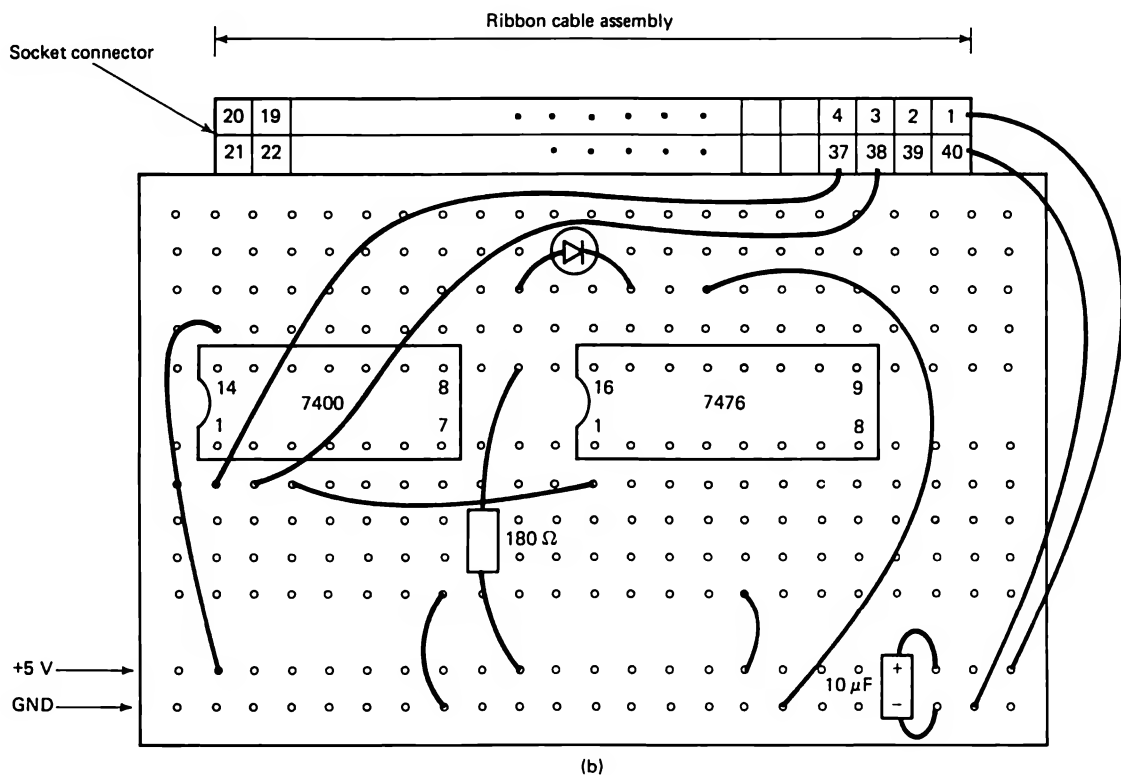
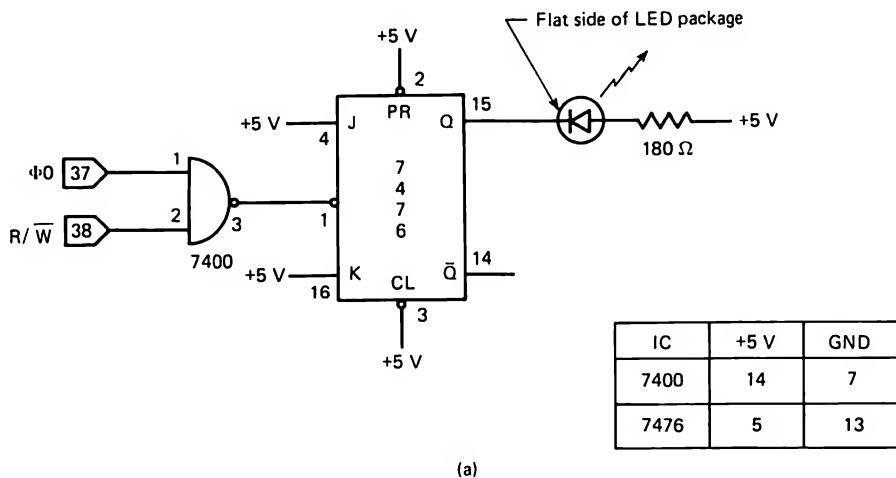


Figure 1-4 Circuit to be wired in step 4 of the experiment: (a) schematic representation; (b) pictorial view of the breadboard and ribbon cable assembly.

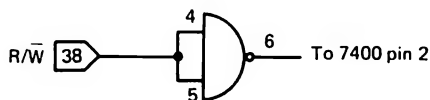


Figure 1-5 The circuit in Fig. 1-4 is modified to monitor memory write cycles by first inverting the R/\overline{W} line and then connecting to the 7400 pin 2 in Fig. 1-4.

Step 6. Try to slow down the program by adding a new line 15.

```
15 FOR I=1 TO 1000: Next I
```

Question 1-2. Why is the LED still continually lit?

Step 7. By inverting the R/\overline{W} line, the circuit in Fig. 1-4 will toggle for *memory write* cycles. Figure 1-5 illustrates the circuit modification. Make this change and then type and run the following program:

```
10 POKE 26128,0
15 FOR I=1 TO 1000: NEXT I
20 GOTO 10
```

Question 1-3. What do you observe?

Note. Although this experiment did not do anything very impressive, we did learn an important lesson. You cannot interface I/O devices to the microcomputer by simply monitoring for read or write machine cycle types. In Experiment 2 we will see how the use of an *address decoder* will allow us to detect special read or write cycles for our I/O device. The 7476 portion of this circuit will be used again in the next experiment.

Step 8. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the “Discussion” and “Procedure” sections.

SOLUTIONS TO QUESTIONS

- 1-1. The memory reads occur so rapidly that your eye cannot detect the LED switching on and off.
- 1-2. The microprocessor is continually accessing memory for operation codes and data (in addition to the PEEK command), causing thousands of memory reads each pass through the program. In fact, because every microprocessor instruction must begin with an op-code fetch (a memory read), the circuit in Fig. 1-4 will always have the LED brightly lit and not slowly switching on and off. This will even be true when the program is not running. Push the RESET key to observe this.
- 1-3. Again the LED is continually lit. Many memory write cycles also occur in a single pass through the program in addition to the memory write due to the POKE command.

EXPERIMENT 2

ADDRESS DECODING

OVERVIEW

In this experiment you will construct an I/O port decoder circuit. Using this decoder, input and output port enable circuits will be built. The effects of partial decoding will also be observed.

OBJECTIVES

The key points to be learned from this experiment are:

1. The microprocessor issues a 16-bit address on address lines A0 through A15 to identify the memory cell or I/O device with which it wishes to communicate.
2. An *address decoder* recognizes one unique address by outputting a pulse when that address is present.
3. An I/O port is enabled by the coincidence of the address select signal and R/\overline{W} pulse.
4. *Partial decoding* results when some of the 16 address lines are not examined by the decoder. In this case, the I/O port will appear to occupy more than one port address.
5. The Apple provides a partially decoded address select signal at each peripheral connector slot.

PARTS LIST

- 1 7432 quad two-input OR gate
- 1 7420 dual four-input NAND gate

- 1 7404 hex inverter
- 1 7476 dual JK flip-flop
- 1 180- Ω resistor (brown-gray-brown)
- 1 LED

DISCUSSION: HARDWARE ADDRESS CONCEPTS

The function of a typical microcomputer is to receive data from its input devices, process this information, and output new data to its output devices. It does this by communicating with memory for specific instructions on where to get the data, what to do with it, and where to output it. Figure 2-1 illustrates this concept and highlights the four main blocks in a typical digital computer.

The memory, I/O ports, and CPU are electrically connected via the three-bus architecture discussed in Experiment 1. The data transfers between the various blocks in Fig. 2-1 occur over the bidirectional *data bus*. The CPU identifies the particular memory location or I/O device by outputting that memory cell's or port's unique address on the *address bus*. In addition, it

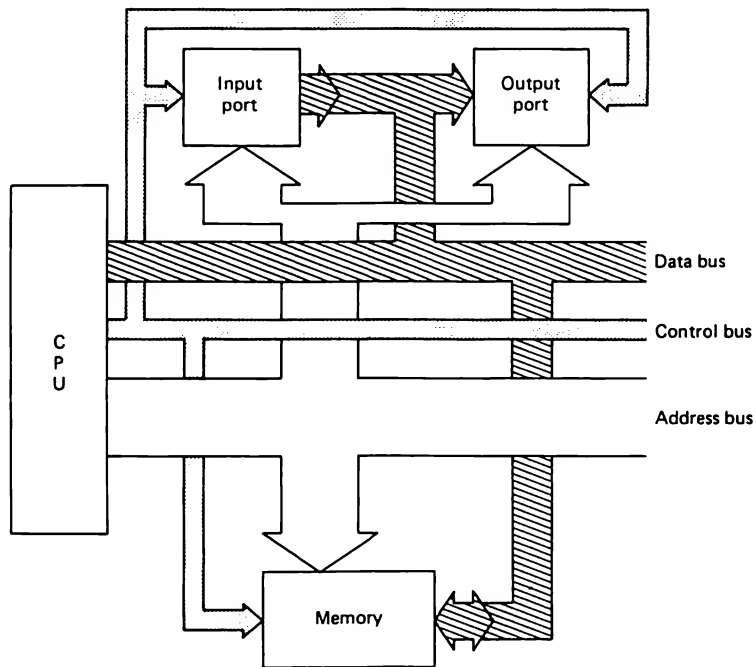
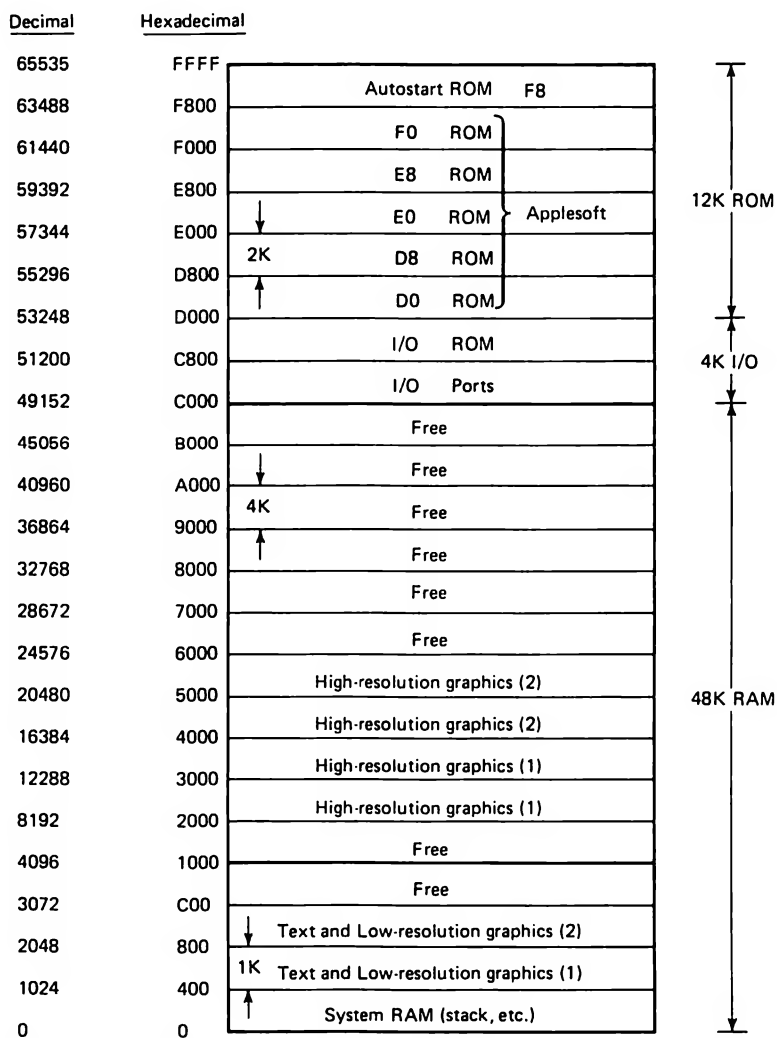


Figure 2-1 Block diagram of a typical digital computer. The CPU fetches instructions from memory, inputs data from its input ports, and outputs data to its output ports.

identifies if the data is entering or leaving the CPU (a read or write operation) by pulsing appropriate lines on the *control bus*.

The address bus for the Apple computer is 16 bits wide (A0 through A15) and is available at each of the eight peripheral connectors (slots 0 through 7). With a 16-bit address there are 65,536 (2^{16}) different addresses possible. This is referred to as *64K* of memory space.

It is convenient to visualize this 64K of memory with a “*map*” detail-



12K ROM

4K I/O

48K RAM

Figure 2-2 Memory map of the Apple II computer. Of the 64K total memory space, 48K is devoted to RAM, 12K to ROM, and 4K to the I/O devices.

ing the addresses of the various types of memory and I/O ports in the computer system. This is shown in Fig. 2-2 for the Apple II.

The left side of this figure details the memory addresses in both *hexadecimal* and *decimal* notations. Notice that all memory locations from location 0 to location 49151 (48K) is devoted to RAM (random access memory). This is read/write memory and is available for user programs, text and graphics screens, and temporary storage for BASIC programs.

Of the remaining 16K of memory space, 12K is used by the six 2K ROM (read-only memory) chips that contain the BASIC interpreter and system monitor. The ROM numbers are identified in the map as D0 through F8. This leaves only the 4K block of memory space between address 49152 and 53247 for all I/O devices.

Figure 2-3 is an enlarged map of this memory space. It is further sub-

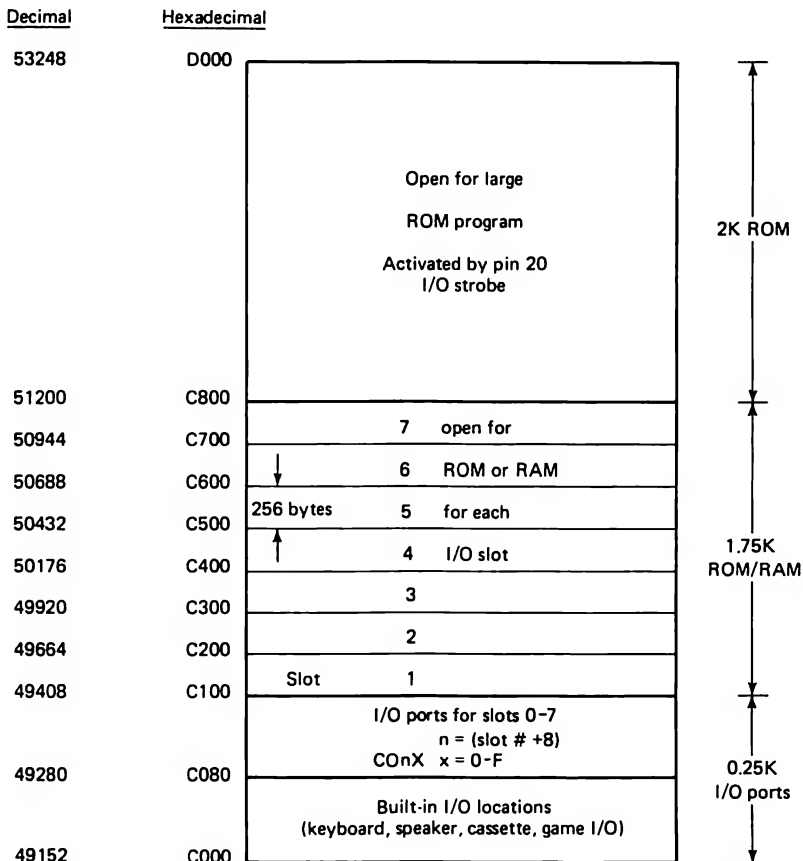


Figure 2-3 Memory map of the 4K of memory reserved for I/O. Normally, only the 128 locations between 49280 and 49407 are used by the peripherals.

divided into two 2K blocks. The top 2K block from 51200 to 53247 is reserved for a large ROM chip on a special-purpose peripheral card. Unless you have such a card, this memory space is unoccupied.

There are seven 256-byte blocks reserved between 49408 and 51199. These are intended for use as an on-card ROM or RAM for special-purpose peripheral cards (printer interfaces, floppy disk controllers, real-time clocks, etc.).

Finally, the remaining 256 bytes between 49152 and 49407 are available for I/O devices. However, the Apple keyboard, speaker, cassette port, and game I/O occupy half of this space, leaving only 128 memory locations available for I/O, shared among the eight slots.

Although it may seem that little space has been left for these I/O devices, any of the memory space from 49280 to 53247 (3968 locations) could in principle be used for I/O if care is taken to avoid conflicts with other peripheral devices. In any case, we will soon see that 128 locations are more than adequate for the majority of I/O applications.

ADDRESS DECODING

A decoder is a circuit that recognizes one particular binary pattern or code. The simplest decoder possible is an *AND gate* (see Appendix D for a review of the various logic gate types). For example, a three-input AND gate will have its output go high only when all three of its inputs are high. It recognizes the binary state 111.

A slight modification to this circuit allows it to decode any binary state. Figure 2-4 illustrates this concept. By selectively adding inverters we may decode any one of the eight possible states.

Because the Apple treats its I/O devices as memory locations (memory-mapped I/O), we must decode the full 16-bit address bus to identify uniquely one I/O port. A practical 16-bit decoder circuit is shown in Fig. 2-5.

Figure 2-6 illustrates a slightly more versatile decoder circuit. By using a 74LS138 3-to-8 line decoder, eight separate address select pulses can be generated. The two 7430 NAND gates require that address lines A15 through A3 equal 1100000010000. If this is true, the 74LS138 will be enabled and

C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Figure 2-4 Simple decoder circuit. Only the input combination 101 will cause the output to go high.

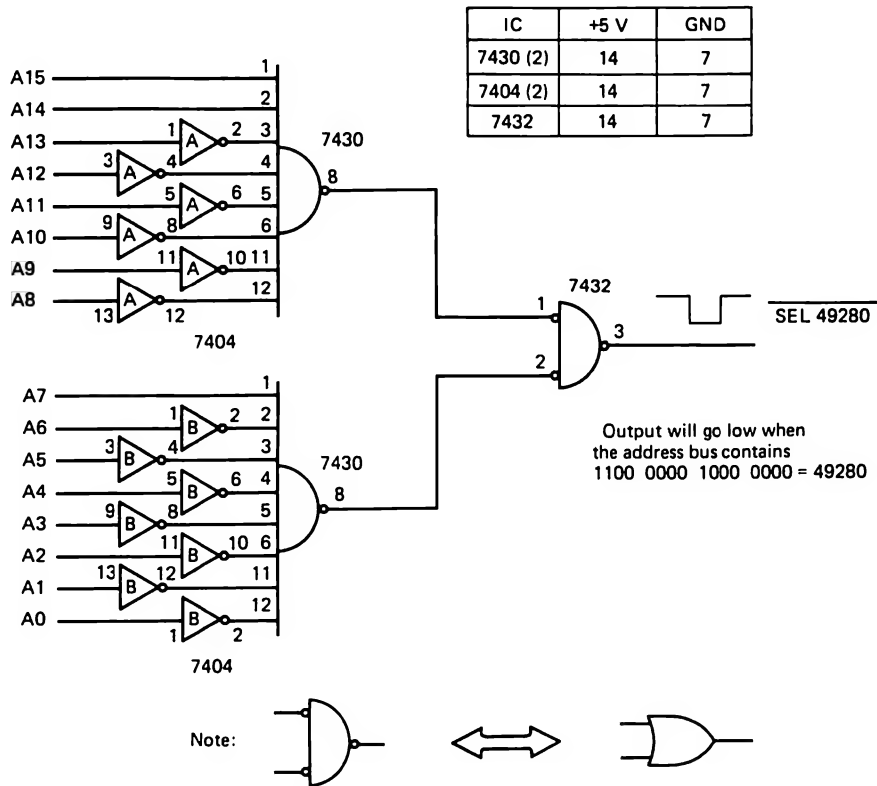


Figure 2-5 Sixteen-bit decoder suitable for memory-mapped I/O on the Apple computer. The binary address is

1100 0000 1000 0000 = 49280

and only this address will activate this circuit.

one of its eight outputs will go low, corresponding to the binary pattern on its ABC inputs (A0 through A2).

Although this circuit is more versatile than the one in Fig. 2-5, it is still quite “messy” to construct, requiring five IC packages and numerous connections. It is for this reason that *partial decoding* is often used.

Partial Decoding

When the decoder does not test all 16 address lines, the resulting circuit is said to be *partially* decoded. An example of such a circuit is shown in Fig. 2-7. The low-order 8 bits of the address bus (A0 through A7) are not examined and therefore become “*don’t cares*.” This means that there will be

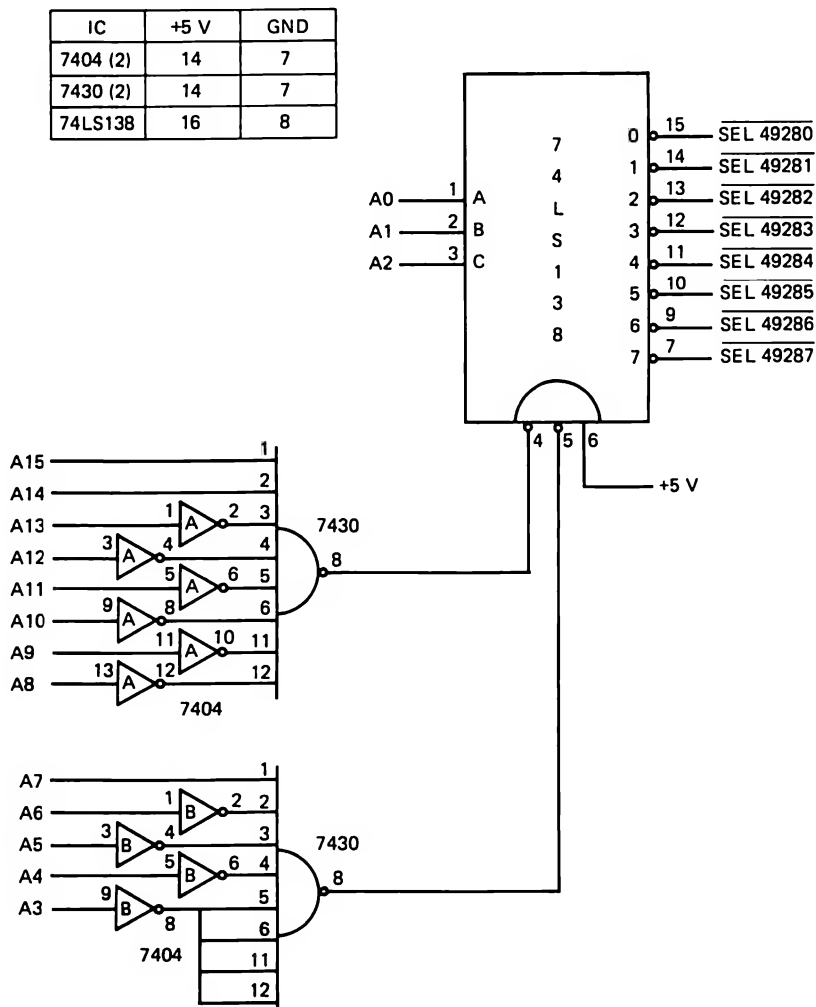


Figure 2-6 This circuit generates eight sequential address select pulses from 49280 to 49287. Each output is active low (goes low when present).

256 different addresses (from 49152 to 49407 in this example) that will all activate this circuit as if they were the same address. These correspond to the 256 combinations of A0 through A7.

The main advantage to partial decoding is the simplification in hardware that results (three fewer chips than the decoder in Fig. 2-5). The main disadvantage is that one port now occupies the memory space of several ports (256 in this case). This can limit system expansion.

IC	+5 V	GND
7404	14	7
7430	14	7

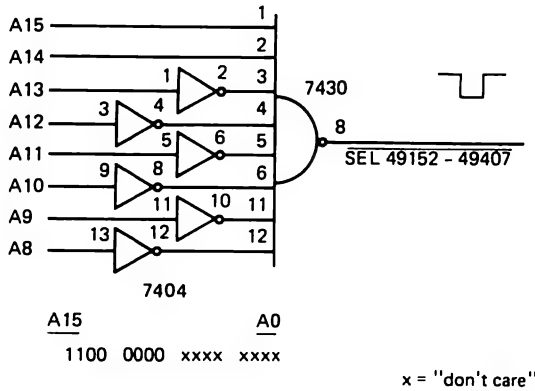


Figure 2-7 Partial decoding results when some of the 16 address lines are not examined by the decoder. These lines then become *don't cares*.

In the case of the Apple II, the circuit in Fig. 2-7 clearly will not work. This is because the memory space between 49152 and 49407, while reserved for I/O devices, must accommodate many I/O devices, not just one.

Fortunately for us, the designers of the Apple have made our job of address decoding somewhat simpler by already decoding the 12 high-order bits of the 16-bit address bus. Figure 2-8 is equivalent to the decoder circuit found in the Apple (the actual circuit is somewhat more complex).

Example 2-1

Refer to Fig. 2-8 and determine the range of addresses for each address select output in this circuit. (*Note:* Apple refers to these outputs as “*device selects*.”)

Solution First examine the 74LS138 enable inputs. The two active-low inputs are driven by the 7430, and in order for this gate's output to be low, its binary input must be 11000000 (A15 through A8). We also see that A7 must be high to enable the 74LS138 decoder. When both of these conditions are true, the ABC inputs (A4 through A6) are decoded to produce the eight device select outputs. For example, the $\overline{\text{DEV4}}$ output will be low when A6 through A4 = 100. The full 16-bit code must then be

$$1100\ 0000\ 1100\ \text{XXXX} = 49344\text{--}49359$$

Because address lines A0 through A3 are not examined, these 4 bits become *don't cares* and each device select line will actually respond to 16 different addresses. These are listed in Table 2-1.

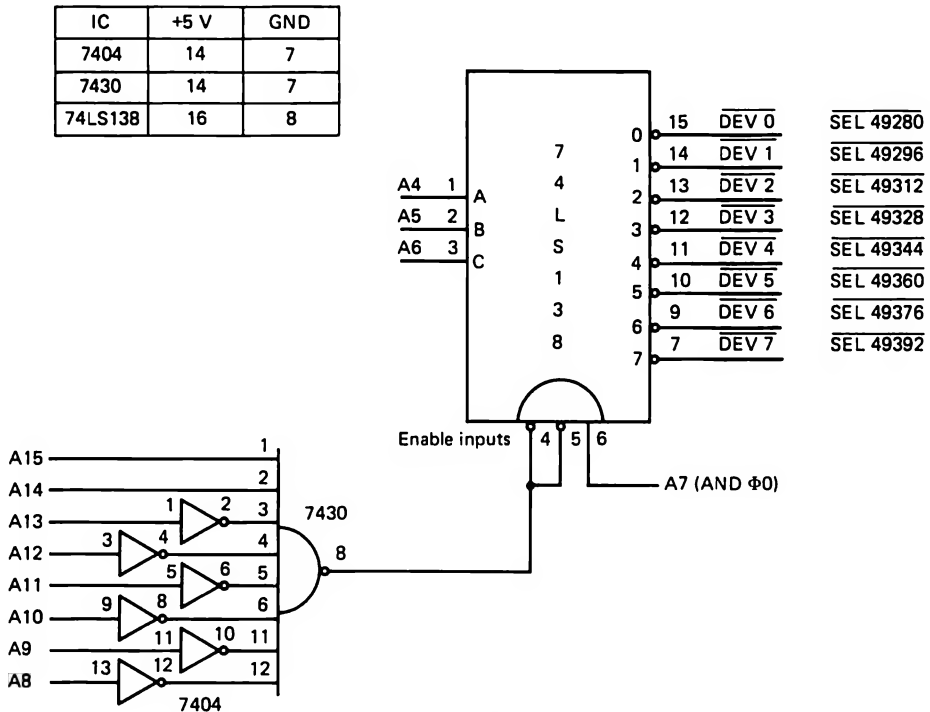


Figure 2-8 Equivalent circuit to the address decoder built into the Apple computer. Each device select pulse is brought to a separate I/O slot (pin 41).

The following points summarize the address decoding features of the Apple computer.

1. All I/O devices must normally reside between addresses 49280 and 49407.
2. Twelve of the 16 address lines are already decoded to this range on the Apple motherboard.
3. Each peripheral slot connector has a signal ($\overline{\text{DEVICE SELECT}}$) which goes low when the address bus contains an address in its range. For example, any address from 49360 to 49375 will cause the $\overline{\text{DEVICE SELECT}}$ at peripheral slot 5 to go low. Table 2-1 summarizes this information.

Example 2-2

Design a circuit to produce a pulse output when address 49399 appears on the address bus. Which slot would normally be used for this circuit?

Solution The circuit is shown in Fig. 2-9. Because address 49399 falls between 49392 and 49407, the proper slot is 7 (see Table 2-1). The low 4 bits must equal a number between 0 and 15. This is calculated as

$$49399 - 49392 = 7 = 0111_2$$

The 7420 decodes these low 4 bits to make this a *fully decoded* circuit.

TABLE 2-1 THE EIGHT DEVICE SELECT OUTPUTS IN FIG. 2-8 ARE NOT FULLY DECODED; EACH OUTPUT CORRESPONDS TO 16 DIFFERENT ADDRESSES

Signal name	Slot number	Range of addresses
$\overline{\text{DEV0}}$	0	49280-49295
$\overline{\text{DEV1}}$	1	49296-49311
$\overline{\text{DEV2}}$	2	49312-49327
$\overline{\text{DEV3}}$	3	49328-49343
$\overline{\text{DEV4}}$	4	49344-49359
$\overline{\text{DEV5}}$	5	49360-49375
$\overline{\text{DEV6}}$	6	49376-49391
$\overline{\text{DEV7}}$	7	49392-49407

I/O Port Enable

Input and output ports must receive an enable pulse telling them when they can latch the data on the data bus (an output port) or when to gate their

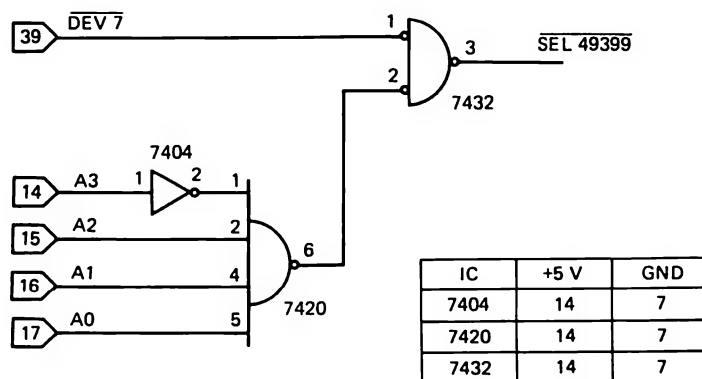


Figure 2-9 This circuit provides full decoding for address 49399 by using the $\overline{\text{DEV7}}$ signal available at peripheral connector 7. The pin numbers shown represent the numbers at the socket connector end of the ribbon cable assembly.

data onto the data bus (an input port). The address select pulse alone cannot do this.

Refer to the timing diagrams in Fig. 1-3. Although the address bus contains a valid address early in the machine cycle, the data bus is not ready to receive or output data until near the end of that cycle. When exactly is it ready? By carefully studying Fig. 1-3, you should be able to see that the $\Phi 0$ clock signal establishes the timing. When $\Phi 0$ is high *AND* R/\overline{W} is *high*, the microprocessor is ready to input data. Conversely, when $\Phi 0$ is high *AND* R/\overline{W} is *low*, the microprocessor presents output data on the data bus. Only when $\Phi 0$ is high does the microprocessor actually use its data bus.

I said earlier that Fig. 2-8 was equivalent to the decoder circuit in the Apple. One point that was purposely left open in that discussion was the active-high enable input to the 74LS138 (labeled A7 *AND* $\Phi 0$). The Apple designers have again simplified our decoding task. Notice how $\Phi 0$ is used in this circuit. Only when A7 *AND* $\Phi 0$ are high can the 74LS138 be enabled. This means that each address select output, $\overline{DEV0}$ through $\overline{DEV7}$, will be active only if their address is valid *AND* $\Phi 0$ is high. Therefore, it will not be necessary to use $\Phi 0$ in our interfaces provided that one of the device select outputs is used. This is because these outputs are already qualified with the $\Phi 0$ clock signal.

Figure 2-10 is a circuit that provides input and output port enable pulses. It uses the full decoder from Fig. 2-9 and develops an $\overline{OUT 49399}$

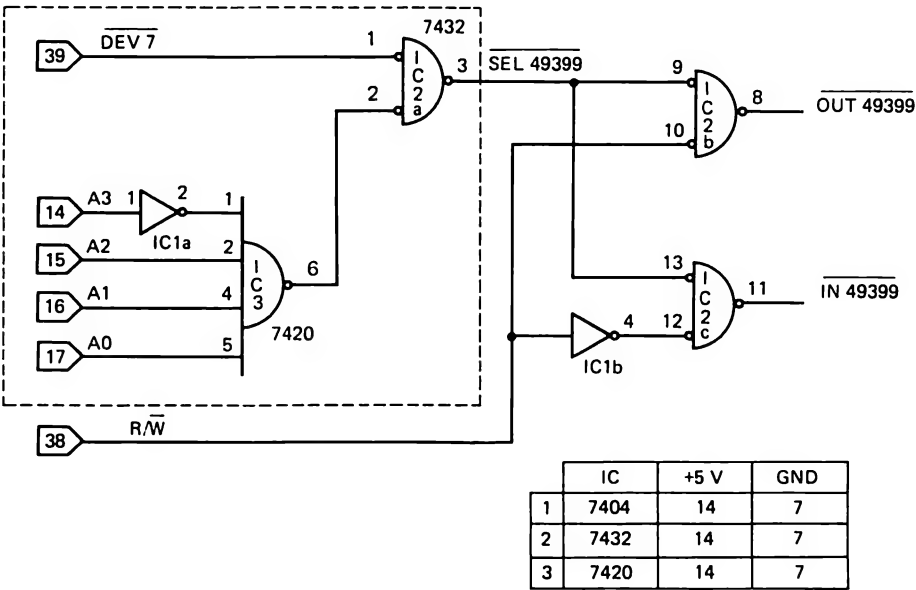


Figure 2-10 The circuit from Fig. 2-9 (enclosed within dashed lines) is used to generate input and output port enable signals.

enable signal and an $\overline{\text{IN 49399}}$ enable signal. Notice how the $\text{R}/\overline{\text{W}}$ line is used to generate these two separate outputs.

In Experiments 3 and 4 we will see how to use these enable signals when adding input and output ports to the Apple.

PROCEDURE

Note. Steps 1 through 3 of Experiment 1 should be followed before attempting this and all other experiments in this book.

Step 1. Refer to Fig. 2-10 and wire this circuit on your breadboard. Three ICs will be required. Be sure that the Vector card is in slot 7.

Note. Never wire a circuit or plug or remove cards from your Apple with the computer power on.

Step 2. Refer to Fig. 2-11 and wire the 7476 JK flip-flop as shown. Connect the clock input (pin 1) to the $\overline{\text{OUT 49399}}$ output (7432 pin 8).

Step 3. Load and run the following program.

```

10 POKE 49399,Y           :REM ADDRESS BUS=49399
20 FOR I=1 TO 1000: NEXT I :REM TIME DELAY
30 GOTO 10

```

Question 2-1. You should have observed the LED to switch slowly ON and OFF. Explain why.

Step 4. Rerun the program of step 3 but use a different address. Now run the program with the correct address but change line 10 to

```
10 Y=PEEK(49399)
```

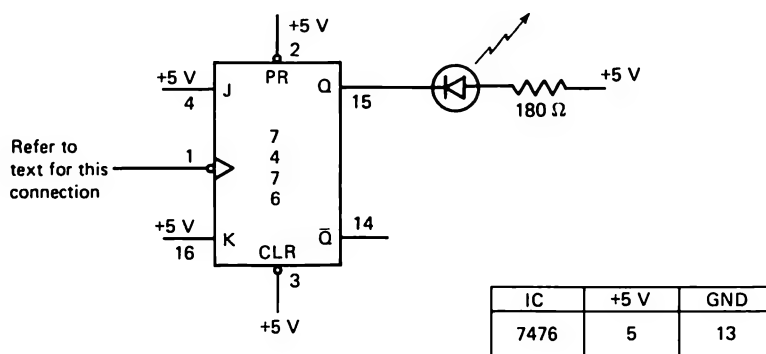


Figure 2-11 JK flip-flop connections to be made in step 2 of the experiment.

Question 2-2. What two signal conditions are required to obtain the $\overline{\text{OUT49399}}$ signal? What *hardware* change is needed to make the PEEK command work?

Note. You may have noticed that the $\overline{\text{IN49399}}$ output pulsed rather unexplicably for both the PEEK and POKE commands. This strange result is a consequence of the 6502 microprocessor's rather simple control bus, consisting of only $\Phi 0$ and $\text{R}/\overline{\text{W}}$. The $\text{R}/\overline{\text{W}}$ line is normally high and pulses low for write machine cycles. When the processor executes internal operations (incrementing the accumulator, for example) the $\text{R}/\overline{\text{W}}$ line is left high. During this time the address bus is not "floated" but left with whatever "old" address was present from a previous instruction. Thus we can have the odd situation where a POKE command establishes an address which is left on the bus. A subsequent *internal* CPU operation causes $\text{R}/\overline{\text{W}}$ to be high and our circuit decodes this and generates an $\overline{\text{IN}}$ pulse. Fortunately, no harm is done. The CPU closes its data bus while performing these internal tasks, effectively ignoring the outside world.

Step 5. Turn off the computer. Now move the Vector card to another slot, say slot 4. Turn on the computer and run the program in step 3, again monitoring the $\overline{\text{OUT49399}}$ output.

Question 2-3. Why doesn't the circuit respond to this program? What changes must be made?

Step 6. Once you feel that you understand how to predict the address associated with each slot, replace the Vector card in slot 7. Now connect $\overline{\text{OUT49399}}$ to the preset flip-flop input: pin 2. Connect $\overline{\text{IN49399}}$ to the clear input: pin 3. Wire the clock input to +5 V.

Step 7. Enter the following program into your computer and run it.

```

10 POKE 49399,Y           :REM ADDRESS BUS=49399, R/ $\overline{\text{W}}$ =0
20 FOR I=1 TO 1000: NEXT I :REM TIME DELAY
30 Y=PEEK(49399)          :REM R/ $\overline{\text{W}}$ =1
40 FOR I=1 TO 1000: NEXT I :REM TIME DELAY
50 GOTO 10

```

The LED should switch on and off.

Question 2-4. Once you feel that you understand the operation of this program, write a new program that asks if you want the LED ON or OFF, and takes the appropriate action (remember that the LED goes ON when the flip-flop Q output goes low). After completing this question you should be able to appreciate that the port enable pulses produced by this circuit could be used for purposes other than enabling I/O ports (for example, turn on a relay, a valve, a motor, etc.).

Step 8. Partially decode the circuit in Fig. 2-10 by connecting $\overline{\text{DEV7}}$ directly to pins 9 and 13 of the 7432 (remove the 7420 and 7432 gate a).

Load and run the following program.

```

10 HOME
20 INPUT "WHAT PORT ADDRESS WOULD YOU LIKE";P
30 FOR J=1 TO 4
40 POKE P,Y
50 FOR I=1 TO 500: NEXT I
60 Y=PEEK(P)
70 FOR I=1 TO 500: NEXT I
80 NEXT J
90 GOTO 10

```

Question 2-5. What range of addresses did you find that would cause the LED to switch ON and OFF?

Step 9. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

SOLUTIONS TO QUESTIONS

- 2-1. Each time the **POKE 49399,Y** command is executed the $\overline{\text{SEL49399}}$ output and $\text{R}/\overline{\text{W}}$ line go low. This causes $\overline{\text{OUT49399}}$ to go low, clocking the flip-flop each time the **POKE** is executed.
- 2-2. The $\overline{\text{OUT49399}}$ output will be activated only if the address bus contains 49399 **AND** $\text{R}/\overline{\text{W}}$ is low (a memory write cycle). Only the **POKE 49399,Y** command will accomplish this. The $\overline{\text{IN49399}}$ output is activated by the **Y=PEEK(49399)** command. Connect the flip-flop clock input to this output (7432 pin 11) instead of pin 8.
- 2-3. When plugged into slot 4, the $\overline{\text{DEV7}}$ line in Fig. 2-9 becomes $\overline{\text{DEV4}}$. Referring to Table 2-1, this means that the new address is 49351 ($49344 + 7$). Line 10 should be changed to **POKE 49351,Y**.
- 2-4. One possible program is

```

10 HOME
20 INPUT "DO YOU WANT TO TURN THE LED ON OR OFF":A$
30 IF A$="OFF" THEN 70
40 IF A$ <> "ON" THEN 20
50 Y=PEEK(49399)      :REM TURN LED ON
60 GOTO 10
70 POKE 49399,Y       :REM TURN LED OFF
80 GOTO 10

```

- 2-5. You should have found that any address from 49392 to 49407 caused $\overline{\text{DEV7}}$ to go low and activate the flip-flop. The binary address is 110000001111XXXX.

EXPERIMENT 3

MEMORY-MAPPED OUTPUT PORT CONCEPTS

OVERVIEW

In this experiment you will wire a 7485 4-bit comparator and a four-position DIP switch as a port decoder that is switch selectable to 1 of 16 addresses. You will use this circuit to enable an output port constructed from a 74100 8-bit latch.

OBJECTIVES

The key points to be learned from this experiment are:

1. A computer output port requires an address decoder, a port enable signal, and a latch.
2. A latch is an essential component in an output port because the output data is on the data bus only for fractions of a microsecond.
3. The **POKE X,Y** command is used to force data Y onto the data bus and address X onto the 16-bit address bus.

PARTS LIST

- 1 7485 4-bit comparator
- 1 7427 triple three-input NOR gate
- 1 74100 8-bit latch
- 1 8-position DIP (dual-in-line package) switch (Radio Shack 275-1301)
- 8 180- Ω resistors (brown-gray-brown)
- 8 LEDs

DISCUSSION: THE POKE X,Y COMMAND

When using Applesoft, data is output to a port with the command **POKE X,Y**. The sequence is as follows:

1. The address of port X is placed on the 16 address lines (A0 through A15).
2. Data Y is placed on the internal bidirectional data bus (now configured as output lines).
3. The R/\overline{W} control line goes low and a valid address appears on A0 through A15. When $\Phi 0$ goes high, data is available on D0 through D7.

Because the data bus is limited to 8 bits, Y is restricted to integers between 0 and 255. Similarly, X can represent any integer address between 0 and 65535. You should be able to see that by using the **POKE X,Y** command we can force any data and address to appear on the data and address buses.

The job the hardware must perform should then be clear. It must:

1. Decode the port address.
2. Use the address select pulse from the decoder and R/\overline{W} pulse to generate a port enable pulse.
3. Use this port enable pulse to enable a *latch* to store the output data.

Typical Output Port

Figure 3-1 illustrates a typical microcomputer 8-bit output port. The 7420 "looks for" a seven on A0 through A3 of the address bus. When this occurs *AND* $\overline{DEV7}$ is low, the address matches. As discussed in Experiment 2 and shown in Table 2-1, the full address is $49392+7 = 49399$. Now, if the R/\overline{W} line is also low, the 7427 output will go high and enable the *latch*. The Q outputs of the 74100 will catch and store the contents of the data bus.

Note that the reason this circuit works is the existence of unique conditions on the three buses when the **POKE 49399,Y** command is executed. No other BASIC command can activate this circuit.

Although this circuit uses a 74100 latch, other, equivalent circuits can be used. Table 3-1 is a partial list of several common latches. Their distinguishing features are listed. Latches such as the 7475 and 74100 are *level triggered*, meaning that the outputs will latch the input data when the clock is at a certain voltage level (0 V in this case). The other latches in the table are *edge triggered* and they will latch the input data the instant the clock edge occurs (the rising edge in this case).

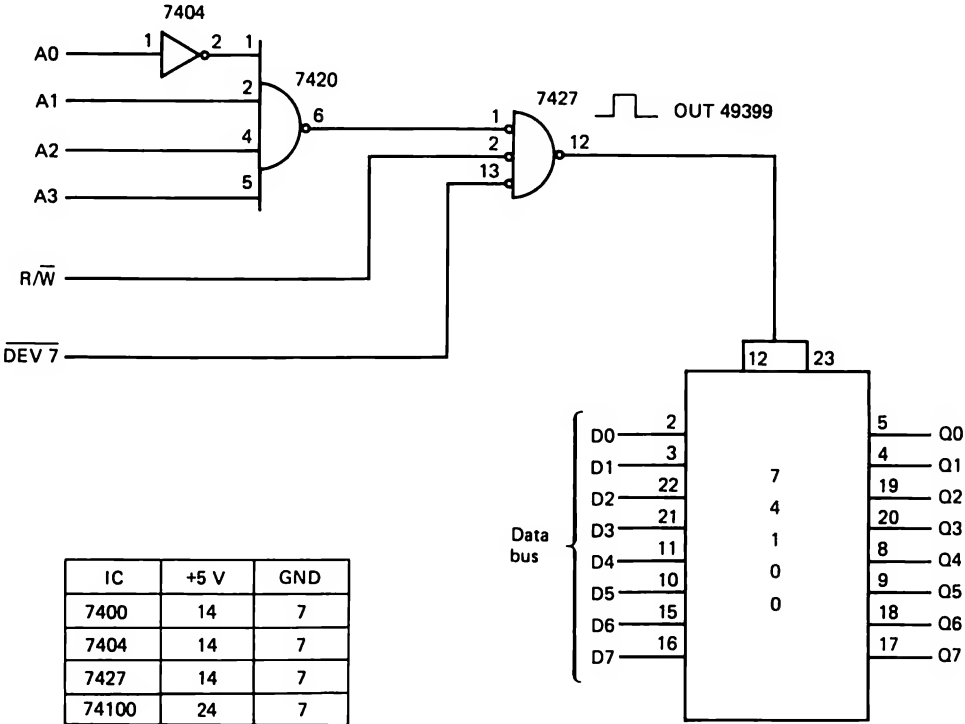


Figure 3-1 This circuit is nearly identical to the decoder circuit in Fig. 2-10. When the address bus holds 49399 AND R/W is low, the 7427 output will be high, the latch enabled, and the contents of the data bus latched at outputs Q0 through Q7.

TABLE 3-1 TYPICAL LATCHES SUITABLE AS MICROCOMPUTER OUTPUT PORTS

Part number	Description	Clock	Package type
7475	4-bit latch	Level triggered	16-pin DIP
74100	8-bit latch	Level triggered	24-pin DIP
74173	4-bit latch with tri-state outputs	↑ Edge triggered	16-pin DIP
74174	6-bit latch	↑ Edge triggered	16-pin DIP
74175	4-bit latch	↑ Edge triggered	16-pin DIP
74LS373	8-bit latch with tri-state outputs	Level triggered	20-pin DIP

The Data Output Is Always 8 Bits

This might already be obvious to you, but there is no way to instruct the microprocessor to output less than 8 bits of data. Of course, we don't have to latch all 8 bits. For example, a very simple 1-bit output port is illustrated in Fig. 3-2. This circuit uses *partial decoding* such that whenever $\overline{DEV7}$ is low *AND* R/\overline{W} is low, the D flip-flop will receive a clock pulse and latch bit DO of the data bus.

Example 3-1

Determine BASIC commands to set and reset the flip-flop in Fig. 3-2.

Solution Any address between 49392 and 49407 will work. Therefore, to set the flip-flop we could use

POKE 49392,1

and to reset the flip-flop we could use

POKE 49392,0

Suppose that the 8-bit output port in Fig. 3-1 is wired on your bread-board and the command POKE 49392,73 is given. What will be seen at Q0 through Q7?

The answer is the binary equivalent of 73, or 01001001. Actually, the numeric value of the data output may not be as important as the *binary pattern* of 1's and 0's that is established. For example, Fig. 3-3 illustrates

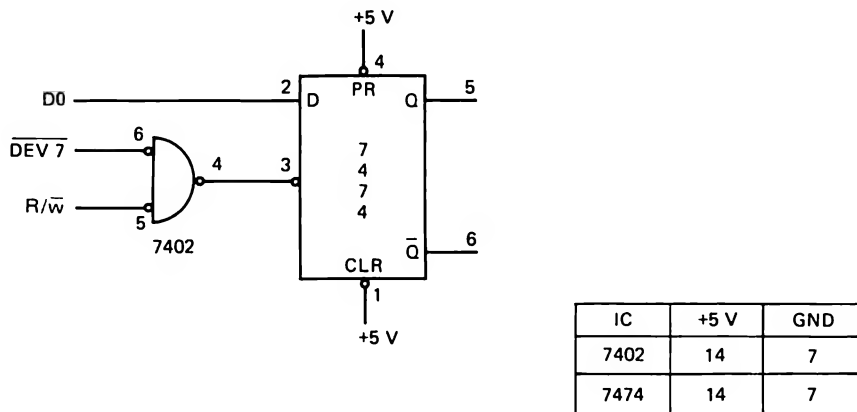


Figure 3-2 One-bit partially decoded output port. This circuit will respond to any port address between 49392 and 49407 and latch data bus bit DO.

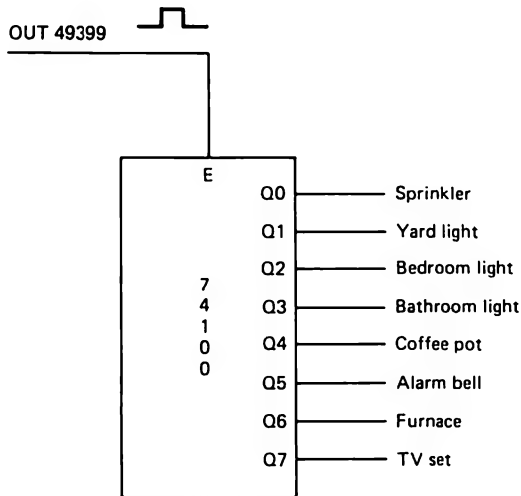


Figure 3-3 If desired, the 8 bits of an output port can be used to control appliances in the home. In this case, a particular binary pattern must be output to activate the proper appliances.

the 8-bit latch from Fig. 3-1 connected to various conveniences in the home. Let us assume that we wish to turn on the yard light, the bathroom light, the coffee pot, and the furnace but to keep the other appliances off. What number should we output to port 49399?

First, write down the desired binary pattern (assuming that a logic 1 will activate the appliance):

01011010

Now converting to decimal: $64 + 16 + 8 + 2 = 90$. The BASIC command should be **POKE 49399,90**.

Seven-Segment Display Output Port

Suppose that we wish to add a display to our computer that can be used to count events in decimal from 0 to 99. A possible circuit is shown in Fig. 3-4. In this circuit we have arbitrarily selected output port 49407 and have chosen two common-anode *seven-segment* displays. The 7475's will latch the data bus contents when A0 through A3 equal 1111, A4 through A15 equal 110000001000 ($\overline{\text{DEV7}}$), AND the R/ $\overline{\text{W}}$ line is low. Each 7475 output is then decoded to provide the proper seven-segment format for the two DL707 displays.

Thinking that all is well, we might proceed to test this circuit by executing the command **POKE 49407,5**. The displays show 05. Next we try **POKE 49407,52**. The displays show 34! What is wrong?

What we have forgotten is that although BASIC "thinks" in decimal, when it outputs to a port, it does so in binary. When 52 is output, the actual

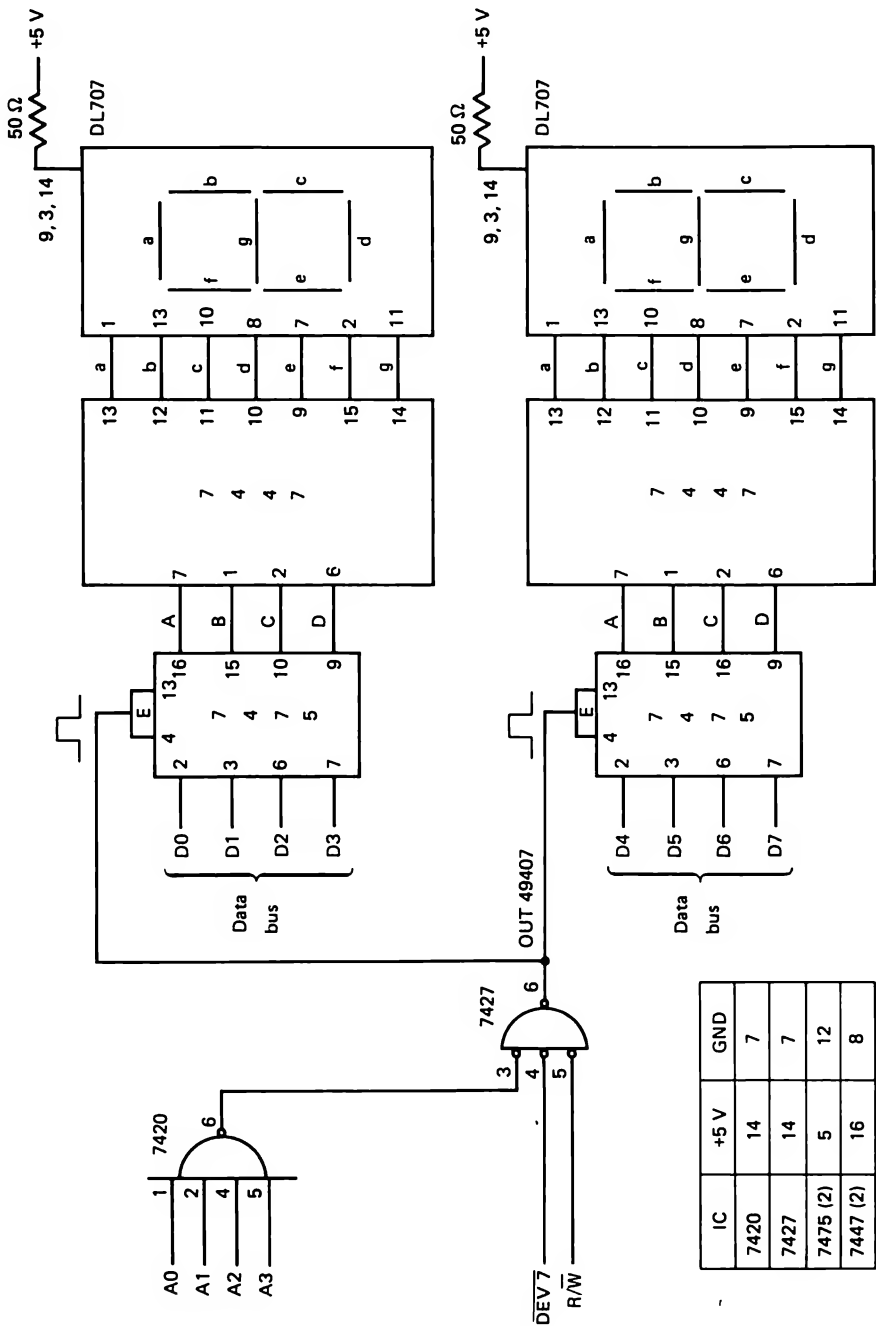


Figure 3-4 Seven-segment display wired to output port 49407 on the Apple.

pattern on the data bus is 00110100. The most significant display receives 0011 (3) and the least significant display receives 0100 (4). Hence we see 34 and not 52.

The solution to this problem is in the software. We must adjust each decimal number before it is output so that it appears correctly in the displays. Realizing that each display receives 4 bits, there will not be a carry into the most significant display until a number greater than 15 (1111) is output. In fact, the least significant display will count units as in decimal, but the most significant display counts 16's instead of 10's. What must be done is to convert the most significant digit to a decimal value, assuming a weight of 16 for its position.

Example 3-2

Convert 52_{10} to the proper format for output to two seven-segment displays.

Solution

1. The least significant display should receive the units' digit without change: in this case, 2.
2. The most significant display should show a 5. Converting to a base of 16 for this digit, we have $5 \times 16 = 80$.
3. Add the two numbers: $80 + 2 = 82$.
4. Check: $82 = 01010010$ or 52 in the two seven-segment displays.

A BASIC program that will properly output to the two seven-segment displays in Fig. 3-4 is listed below. Note that numbers greater than 99 cannot be displayed and are therefore not allowed.

```

10 HOME
20 INPUT "ENTER ANY INTEGER BETWEEN 0 AND 99";N
30 IF N>99 OR N<0 THEN 20
40 N1=N-(INT(N/10)*10)
50 REM N1=LEAST SIGNIFICANT DISPLAY
60 N2=INT(N/10)*16
70 REM N2=MOST SIGNIFICANT DISPLAY
80 POKE 49407,N1+N2
90 GOTO 20

```

As a final note, those of you who are up on your *hexadecimal* will note that the circuit in Fig. 3-4 actually displays the hexadecimal equivalent of the decimal number input. For example, when 52_{10} is input, the displays show 34, which is the hexadecimal equivalent of 52.

Now if you are also up on your seven-segment decoders, you will know that the 7447 will not display the hex characters A through F. If you are interested in building a hexadecimal display for your computer, replace the two 7475's and 7447's with 9368's. This chip provides a 4-bit latch and hex-

adecimal decoder in one package. You will need to use common-cathode displays, however, because the 9368 outputs are active high.

If you have a little more money to spend and wish to reduce the amount of wiring, use a *TIL311* or *HP 5082-7340* display. These chips have the latch, hexadecimal decoder, and LEDs all in one package. All that is needed are the power connections and the four binary inputs.

Refer to Experiment 11, Question 11-5, for a software-only technique for driving a seven-segment display. This method does away completely with the seven-segment decoder chips.

PROCEDURE

Step 1. Study the circuit in Fig. 3-5. Now carefully wire it on your breadboard, noting the following:

1. Mount the DIP switch *upside down* on the breadboard with switch 1 to your right. In this way, when the switch handle is down it will correspond to a 0, and to a 1 when up.
2. Be careful to orient all LEDs as shown (flat side of the package to the resistor).

Note. The 7485 is a 4-bit *magnitude comparator*. When word A equals word B, its A=B output will go high. For this to happen, the switch settings (S1 through S4) will have to be matched by A0 through A3 of the address bus. In addition, the high-order address bus must cause $\overline{DEV7}$ to be active. When both of these conditions are met *AND* R/\overline{W} is low, the 7427 output will pulse high and the 74100 will be enabled. The range of port addresses is 49392 to 49407.

Step 2. Set the DIP switch to all 0's. Test your hardware with the following in the immediate mode.

POKE 49392,0

Question 3-1. What should you observe? How do you turn all the LEDs OFF?

Step 3. Set all switches to the 1 (up) position. Again try outputting to port 49392 and turn the LEDs ON or OFF. You should not be successful because the port code is no longer 49392. Use the proper port address in the POKE command and try again. You should now be able to turn the LEDs ON or OFF.

Question 3-2. Write a BASIC program that asks you for the port address and the specific LEDs you want turned on. Run the program and verify proper operation. Remember that an ON LED requires a 0 output. Don't

forget to set the switches to the port value you have chosen.

Step 4. Load and run the following program with all switches in the 0 position.

```

10 FOR J=255 TO 0 STEP -1
20 POKE 49392,J
30 FOR I=1 TO 100: NEXT I
40 NEXT J
50 GOTO 10

```

Question 3-3. What happens if line 10 is changed to the following?

```

10 FOR J=0 TO 255

```

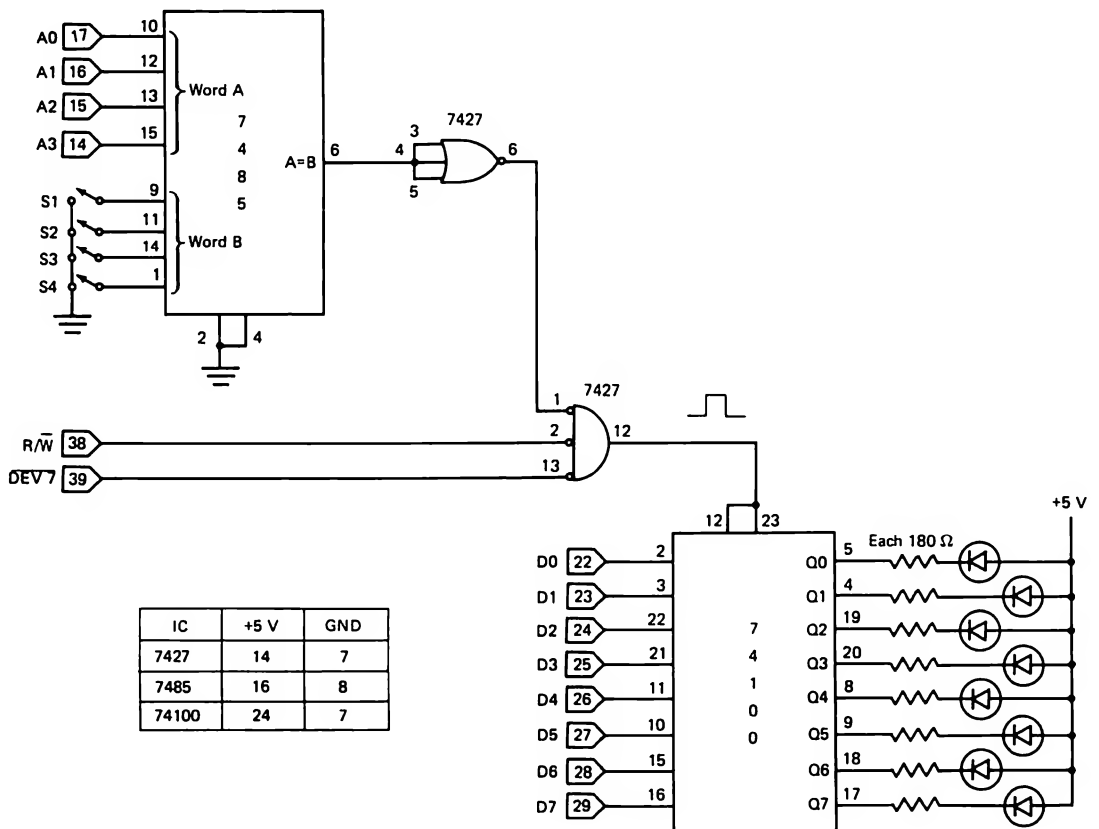


Figure 3-5 Schematic diagram of an 8-bit output port switch selectable to one of 16 different port addresses.

Step 5. Write a BASIC program similar to the seven-segment display program at the end of the "Discussion" section. This program should input a number between 0 and 99 and output to the LEDs. You will have to interpret the LEDs as two groups of four (Q0 through Q3 and Q4 through Q7) to verify program operation.

Note. Because a logic 1 will turn an LED OFF, a lit LED will actually correspond to a 0, not a 1. This problem can be corrected by outputting the complement of the desired number (255-n).

Step 6. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

SOLUTIONS TO QUESTIONS

3-1. 0 = 00000000; all LEDs are ON. The command POKE 49392,255 will turn all LEDs OFF.

3-2. One possible solution:

```
10 HOME
20 L=0           :REM L IS DATA TO BE OUTPUT
30 INPUT "ENTER THE PORT ADDRESS (49392-49407) ";P
40 IF P>49407 OR P<49392 THEN 30
50 FOR J=0 TO 7
60 PRINT "TURN LED ";J;" ON/OFF ";
70 INPUT L$
80 IF L$="OFF" THEN L=L+(2^J)
90 NEXT J
100 POKE P,L
110 GOTO 10
```

3-3. Because a 0 output turns ON an LED, the LEDs will count backward when line 10 is changed.

Note. You may want to try other counting programs to cause various effects in the LEDs (for example, one LED appearing to bounce up and down).

EXPERIMENT 4

MEMORY-MAPPED INPUT PORT CONCEPTS

OVERVIEW

In this experiment you will wire an eight-position DIP switch as an input port to the Apple using a 74LS244 octal tri-state buffer. Software techniques will be demonstrated that allow individual switches to be monitored.

OBJECTIVES

The key points to be learned from this experiment are:

1. Data from an input port must be gated onto the bidirectional data bus through *tri-state gates*.
2. The enable pulse for the tri-state gates is generated by the coincidence of the address select pulse *AND* the R/\overline{W} control signal.
3. The command $Y=PEEK(X)$ is used to gate input data onto the data bus from port X and to store it as variable Y.

PARTS LIST

- 1 7404 hex inverter
- 1 7432 quad two-input OR gate
- 1 74LS244 octal buffer (Radio Shack 276-1941)
- 1 8-position DIP (dual-in-line package) switch (Radio Shack 275-1301)
- 8 1-k Ω resistors (brown-black-red)

DISCUSSION: THE **Y=PEEK(X)** COMMAND

Data is input to the Apple from memory or an input port with the BASIC command **Y=PEEK(X)**. When this command is executed, the following sequence of events takes place:

1. The address of port X is placed on the 16 address lines A0 through A15.
2. The bidirectional data bus internally configures itself for data input.
3. The R/\bar{W} control line pulses high and when $\Phi 0$ goes high, the data bus is ready to input data.

As was true with the output port, the data input (Y) is limited to 8 bits, restricting the input data to integers between 0 and 255. The port address, representing a 16-bit code, is restricted to integers between 0 and 65535. In order to input data to the Apple our hardware must:

1. Decode the port address.
2. Use the address select pulse from the decoder and R/\bar{W} pulse to generate an input port enable pulse.
3. Use this enable pulse to gate data onto the data bus lines through *tri-state gates*.

You might have thought that we could connect the data directly to the data bus lines. The problem with this is that the input data would try to hold the data bus lines at specific logic levels, interfering with normal computer operation.

Figure 4-1 illustrates how the tri-state gate solves this problem. Only when enabled is the input logic level passed through the gate onto the bus. In fact, tri-state gates allow many different input ports to be connected to the same bus. The key to this technique is that each set of gates is enabled by a separate address, thereby allowing only one set to drive the bus at a given time.

Tri-state gates are also used to increase the output drive capabilities of the microprocessor. If this is not done, the many circuits connected to the address and data bus may severely affect the processor's ability to communicate reliably with memory and I/O devices. The tri-state gate is called a *buffer* in this application.

Typical Input Port

Figure 4-2 illustrates an 8-bit microcomputer input port for the Apple computer. In this example, *partial decoding* is used such that whenever $\overline{DEV7}$ is

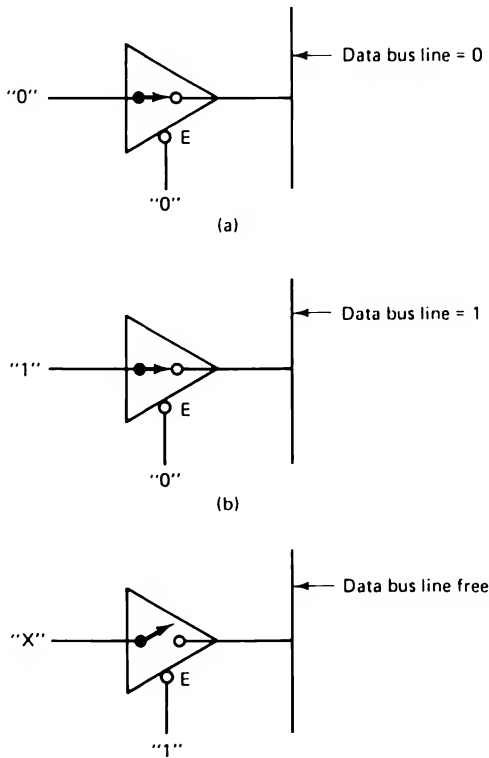


Figure 4-1 Tri-state gates are commonly used to gate data onto a data bus. In (a) and (b) the tri-state gate is enabled and the data bus line is connected to the input logic level. In (c) the tri-state gate is disabled and its output appears as an *open circuit*. The data bus line is now free to be controlled by another driver on the line.

low $AND\ R/\overline{W}$ high, the 7432 OR gate output will go low, enabling the eight tri-state gates. The microcomputer can now read the data established by switches S1 through S8 (simulating eight separate input channels). **The only BASIC command that will activate this hardware is $Y=PEEK(49392)$.***

The circuit in Fig. 4-2 uses the 74LS244 *octal buffer*, but other tri-state circuits can be used for input ports. Figure 4-3 lists several common types. Note that the 74LS241 and 74LS244 are particularly useful with microcomputers because they contain eight gates in one 20-pin package.

The Data Input Is Always 8 Bits

As discussed in Experiment 3, the microcomputer always works with 8 bits of data. In the case of an input port, this means that 8 bits of data will always be read even if the input port itself supplies fewer than eight lines. What will the computer "see" if there are unused input lines? That is a good question, but there can be no specific answer. Generally, a line with no connection to it will be interpreted as a logic 1. This means that when the com-

*Because of the partial decoding, any address from 49392 to 49407 can be used.

IC	+5 V	GND
7404	14	7
7432	14	7
74LS244	20	10

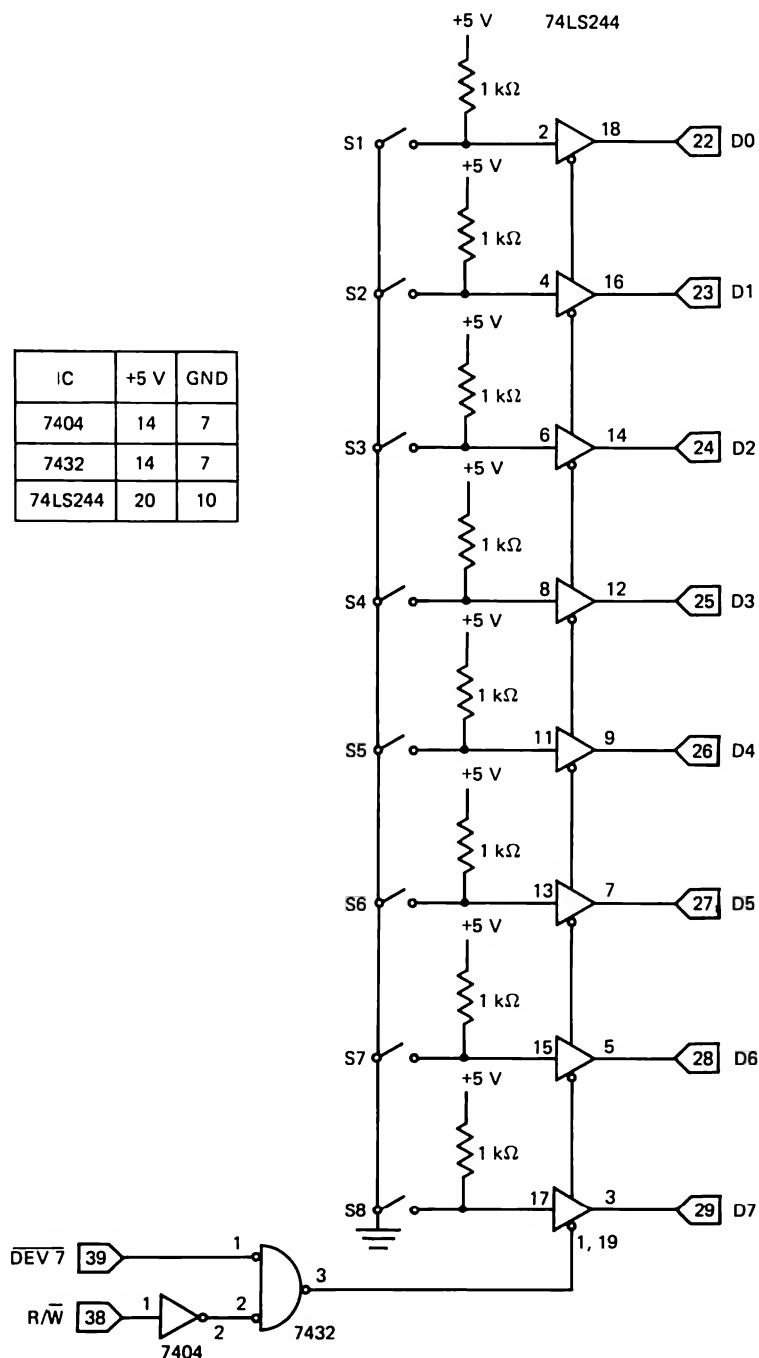


Figure 4-2 Apple computer input port circuit. The port address is determined by $\overline{DEV7}$ and in this case corresponds to any address between 49392 and 49407. When the tri-state gates are enabled, the data byte established by S1 through S8 is gated onto the data bus.

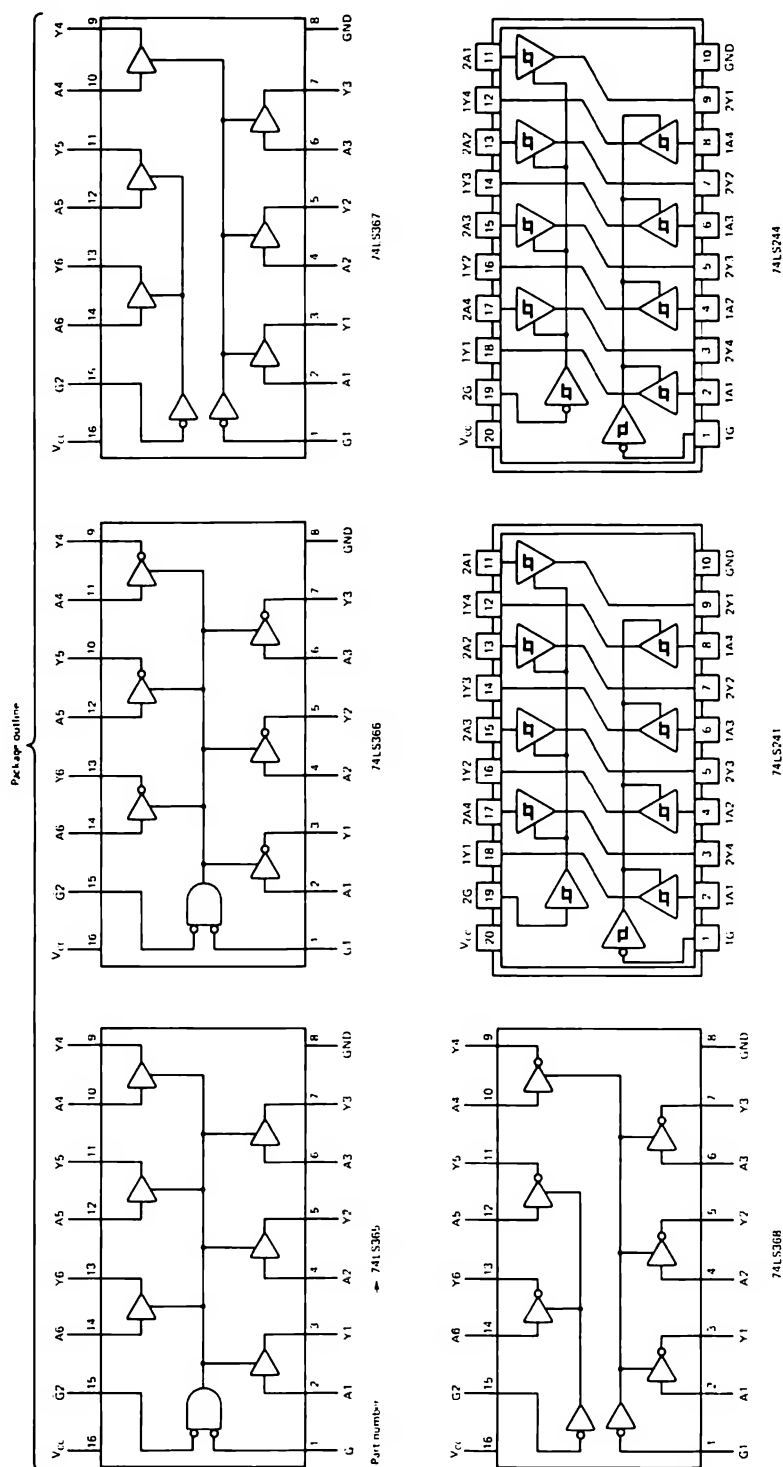


Figure 4-3 Typical tri-state buffers suitable as microcomputer input ports.

mand `Y=PEEK(49392)` is given, and there is no port 49392, the result will probably be `Y=255` or `11111111`. However, digital circuits can be very unpredictable when connected to open circuits and may not always interpret an open line as a 1 [particularly metal oxide semiconductor (MOS) circuits].

The solution to this problem is to use software to test individual bits selectively, ignoring those that are not used. We will explore this technique in the next section.

Masking

The most common method of bit testing is to use a “*masking*” technique. A mask word is chosen and then *ANDed* with the input data. For example, suppose that we wish to test bit 0 of an 8-bit input word. Assume that the input data is `10110011` and we wish to know if bit 0 is a 1. The mask word for testing bit 0 is `00000001`. Note what happens when we *AND* these two words together.

$$\begin{array}{r} 10110011 \quad (\text{input data}) \\ . 00000001 \quad (\text{mask}) \\ \hline 00000001 \end{array}$$

The result of the bit-by-bit *AND* operation is to cause bit 0 of the result to be a 1 if bit 0 of the data is a 1. In general, a *nonzero result* means that the desired bit is a 1, a zero result means that it is not. Any bit can be tested by this method. For example, to test bit 5 the mask should be `00100000`.

Unfortunately, Applesoft does not support a bit-by-bit *AND* operation. Instead, if the two numbers being *ANDed* are nonzero, the result will be a 1; if either or both numbers are zero, the result will be zero.

The only way to use the masking technique with the Apple is to go to the 6502 processor’s native machine language. From BASIC this can be awkward. A `USR` routine can be poked into memory by BASIC, but for our purposes we will use a simpler method.

Converting Decimal to Binary

When inputting data with the `PEEK` command, we obtain a decimal number representative of the binary data at the input port. If we convert this data back to binary, it will be a simpler matter to test individual bits and ignore those not connected.

A decimal number can be converted to binary in several ways. One of the simplest is repeatedly to subtract the highest remaining power of 2 until the result is zero. The binary number may then be written as the concatenation of these binary digits. Appendix C details this method.

Example 4-1

Write a BASIC program to convert any decimal number between 0 and 255 to binary.

Solution The program is given below and uses the *repeated subtraction* method just discussed. The decimal number is assumed to be the variable N, and the result is stored in an array B(J) with B(0) equal to the least significant bit.

```

100 FOR J=7 TO 0 STEP -1      :REM J COUNTS BITS
110 B(J)=1                    :REM ASSUME THE BIT IS A 1
115 REM NOW MAKE B(J)=0 IF N-2^J<0
120 IF N-2^J<0 THEN B(J)=0: GOTO 140
130 N=N-2^J                   :REM SUBTRACT POWER OF 2 IF N>2^J
140 NEXT J

```

Note. If you want to test this program, add an input line,

```
10 INPUT N
```

and an output line,

```
150 FOR J=7 TO 0 STEP -1:PRINT B(J);:NEXT J
```

Now let us suppose that the 8 bits of data we input from a port correspond to various sensors within your home. This is illustrated in Fig. 4-4. Now we would like to write a BASIC program that will determine if the furnace is on or off. Because the furnace sensor is connected to bit D1, we must detect if this bit is high or low.

We can incorporate the routine given in Example 4-1 into a *subroutine* and use this to test bit B(1). For example, if the input port is 49392:

```

10 HOME
20 N=PEEK(49392)              :REM SAMPLE SENSORS
30 GOSUB 100
40 IF B(1)=1 THEN A$="ON ": GOTO 60      :REM TEST BIT 1
50 A$="OFF"
60 VTAB(10): PRINT "THE FURNACE IS: ";A$
70 GOTO 20
100 FOR J=7 TO 0 STEP -1
110 B(J)=1
120 IF N-2^J<0 THEN B(J)=0: GOTO 140
130 N=N-2^J
140 NEXT J
150 RETURN

```

It is also possible to use this technique to test for various combinations of bits. For example, suppose that we wish to know if the front door, kitchen

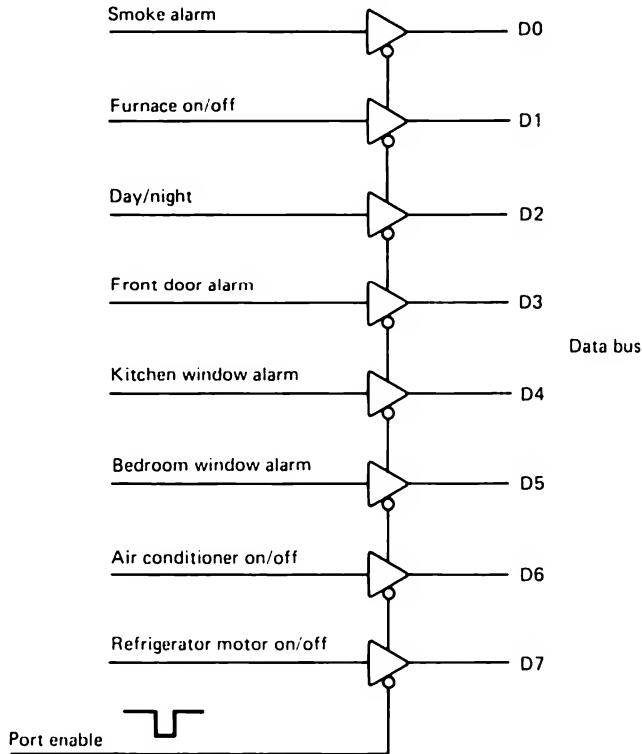


Figure 4-4 The 8 bits of a typical input port may represent conditions within the home. The electronics associated with each sensor is not shown (see Experiment 7).

window, or bedroom window sensors are active. Referring to Fig. 4-4, this translates to bits D3 or D4 or D5 being high. We need only change lines 40 and 50 of the previous program:

```
40 IF B(3) OR B(4) OR B(5) = 1 THEN A$="ALARM ": GOTO 60
50 A$="NO ALARM"
```

PROCEDURE

Step 1. Refer to Fig. 4-2 and carefully wire this circuit on your breadboard. Position the DIP switch *upside down* with switch 1 on your right. In this way a switch in the up position will correspond to a logic 1.

Step 2. Test your circuit by loading and running the following program.

```
10 HOME
20 Y=PEEK(49392)
30 PRINT Y
40 GOTO 20
```

Question 4-1. Explain the result obtained in step 2.

Question 4-2. Modify the program in step 2 so that it prints the following message: THE CURRENT VALUE OF THE SWITCH IS: XXX. Make your modification such that the message is printed only once for any switch setting.

Step 3. Load the following program into your computer. Try both positions for all eight switches for a given bit test. Experiment with the bit tested to be sure that you understand this program.

```
10 HOME
20 INPUT "WHICH BIT DO YOU WANT TO TEST (0-7) ";I
30 N=PEEK(49392)
40 GOSUB 100
50 IF B(I)=1 THEN A$="ON ": GOTO 70
60 A$="OFF"
70 VTAB(12): HTAB(13): PRINT "SWITCH ";I;" IS ";A$
80 VTAB(14): HTAB(6): PRINT "THE BINARY WORD IS: ";
90 FOR J=7 TO 0 STEP -1: PRINT B(J);: NEXT J: GOTO 30
100 FOR J=7 TO 0 STEP -1
110 B(J) = 1
120 IF N-2^J < 0 THEN B(J) = 0: GOTO 140
130 N=N-2^J
140 NEXT J
150 RETURN
```

Question 4-3. Write a program to detect when bits 6 AND 3 = 1 AND when bits 0 OR 4 OR 7 = 1. When the condition is true, the program should display "OK"; if not true, the screen should be blank.

Question 4-4. Write a program to provide a *status report* of all 8 bits of input port 49392 (simulated by the eight-position DIP switch). This program should continually update the display, providing an "instantaneous" status of all 8 bits.

Step 4. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

SOLUTIONS TO QUESTIONS

4-1. You should see the decimal value of the switch setting rapidly scrolling by on the screen. As the switch settings change, the value on the screen will also change.

4-2. One possible way is

```
10 HOME
20 Y=PEEK(49392)
30 PRINT "THE CURRENT VALUE OF THE SWITCH IS: ";Y
40 IF PEEK (49392)=Y THEN 40
50 GOTO 10
```

4-3. One possible solution is

```
10 HOME
20 N=PEEK(49392)
30 GOSUB 100
40 VTAB(12):HTAB(19)
50 IF B(6) AND B(3) AND (B(7) OR B(4) OR B(0))=1 THEN PRINT
   "OK": GOTO 20
60 GOTO 10
100 FOR J=7 TO 0 STEP -1
110 B(J)=1
120 IF N-2^J<0 THEN B(J)=0: GOTO 140
130 N=N-2^J
140 NEXT J
150 RETURN
```

4-4. One possible solution is

```
10 HOME
20 N=PEEK(49392)
30 GOSUB 100
40 FOR J=0 TO 7
50 A$="OFF"
60 VTAB(4+2*J)
70 IF B(J)=1 THEN A$="ON "
80 PRINT "SWITCH ";J;" IS: ";A$
90 NEXT J
95 GOTO 20
100 FOR J=7 TO 0 STEP -1
110 B(J)=1
120 IF N-2^J<0 THEN B(J)=0: GOTO 140
130 N=N-2^J
140 NEXT J
150 RETURN
```

EXPERIMENT 5

INTERFACING THE 8255 PROGRAMMABLE PERIPHERAL INTERFACE

OVERVIEW

In this experiment you will wire an 8255 *programmable peripheral interface* (PPI) to the Apple. You will learn the software techniques needed to initialize the PPI for *mode 0* operation and a traffic light controller utilizing this circuit will be designed and demonstrated.

OBJECTIVES

The key points to be learned from this experiment are:

1. The PPI is a versatile interface circuit that can be programmed to provide several different combinations of input and output ports.
2. This chip meets all the requirements for microcomputer input or output ports without requiring external tri-state gates or latches.
3. The 8255 contains three separate data ports which may be programmed as input or output ports. A fourth port is used to control the mode of operation of the three data ports and is called the *control port*.

PARTS LIST

- 1 8255 programmable peripheral interface (Jameco DP8255)
- 2 7404 hex inverters
- 1 7476 dual JK flip-flop
- 2 green LEDs

- 2 red LEDs
- 2 yellow LEDs
- 6 180- Ω resistors (brown-gray-brown)
- 2 1-K Ω resistors (brown-black-red)
- 2 pushbutton normally open switches

DISCUSSION: A "SMART" I/O DEVICE

The 8255 is a *programmable peripheral interface* (PPI) integrated circuit. It has three I/O ports (or 24 I/O pins) that may be programmed by a microcomputer to be any combination of input and output ports. In fact, port C can be split in half, with four of the pins programmed as inputs, and the remaining four as outputs.

A block diagram of this chip is shown in Fig. 5-1. The three I/O ports can be seen (labeled ports A, B, and C) as well as two control blocks called Group A Control and Group B Control. A unique feature of the 8255 is that the microcomputer can communicate with these two control blocks and tell the PPI what I/O configuration is desired. For example, we might specify port A to be an input port, port B to be an output port, the high-order bits of port C to be inputs, and the low-order bits of port C to be outputs. All that is required is to send the proper *control word* to the two control blocks (actually, the control port) within the 8255. No rewiring is required nor is there a need to supply external tri-state gates and latches, as these are all provided in the 8255.

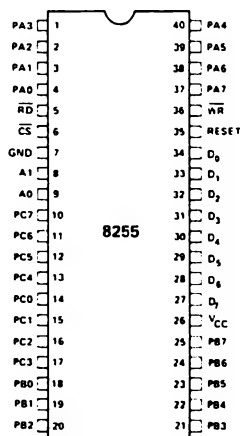
Referring to Fig. 5-1, the 8255 is a 40-pin circuit and requires a single +5-V power source.

Modes of Operation

The 8255 may be programmed to operate in one of three modes, referred to as mode 0, mode 1, and mode 2. Mode 0 provides for basic input and output operations with the three ports A, B, and C. This is referred to as *unconditional* I/O because the microcomputer inputs and outputs data without regard for the I/O port's BUSY/READY status. None of the interface circuits we have discussed thus far has supplied this status line. The I/O ports have always been assumed to be ready.

In some cases, however, the I/O device may have a special status line to indicate that it is busy right now and cannot accept or provide data. When interfacing to this type of device, the microcomputer must check the BUSY/READY flag, and attempt to input or output data only when the I/O device indicates that it is ready. This particular mode of operation is also accommodated by the 8255 and is referred to as *mode 1* or *strobed I/O*. In this mode ports A and B are used for the data paths, but port C is used to

PIN CONFIGURATION



PIN NAMES

D ₇ -D ₀	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
CS	CHIP SELECT
RD	READ INPUT
WR	WRITE INPUT
A ₀ , A ₁	PORT ADDRESS
PA ₇ -PA ₀	PORT A (BIT)
PB ₇ -PB ₀	PORT B (BIT)
PC ₇ -PC ₀	PORT C (BIT)
V _{cc}	+5 VOLTS
GND	0 VOLTS

8255 BLOCK DIAGRAM

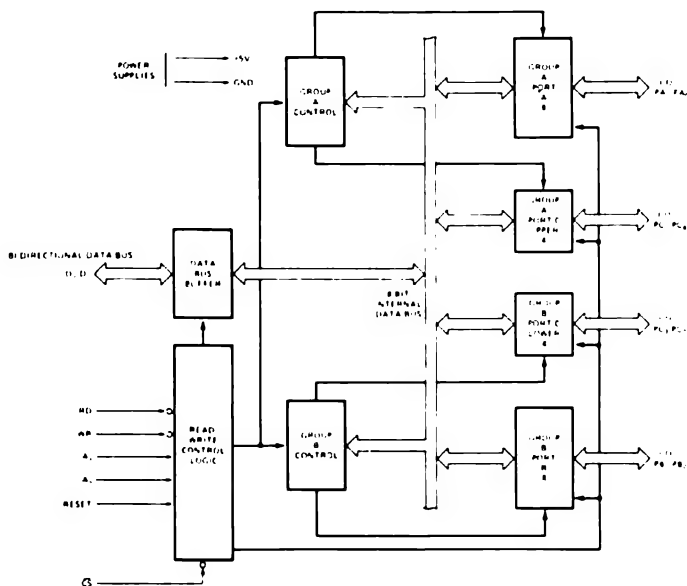


Figure 5-1 Block diagram of the 8255 programmable peripheral interface (PPI). This chip contains three separate I/O ports which may be programmed as inputs or outputs by the microcomputer. (Courtesy of Intel Corporation.)

generate or accept the **BUSY/READY** or “*handshaking*” signals. We will examine this type of I/O in detail in Experiment 11.

Mode 2 is the third mode of operation and is similar to mode 1. The difference is that the data path consists of one set of I/O lines (port A) which are now configured to be *bidirectional*. Port C again provides handshaking capabilities. This mode of operation is intended for I/O devices that communicate over a single 8-bit bidirectional data bus.

In addition to the three modes of operation discussed, there is also a bit set/reset mode. This mode allows individual bits of port C to be set or reset by choosing different control words.

In this experiment we will use only mode 0. Experiment 11 offers more details on mode 1. You are also referred to the 8255 data sheets in Appendix G and available from Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051. Another good reference is *Microcomputer Interfacing with the 8255 PPI Chip* by Paul Goldsbrough (Indianapolis, Ind.: Howard W. Sams and Co., 1979).

Interfacing to the 8255

When the 8255 is interfaced to the Apple, all three system buses must be used. The eight data bus lines are connected directly to the 8255, as shown in Fig. 5-2. The $\overline{RD}/\overline{WR}$ control line is connected to pin 6 (\overline{WR}) and inverted and applied to pin 5 (\overline{RD}). The inverter is necessary to generate an active-low

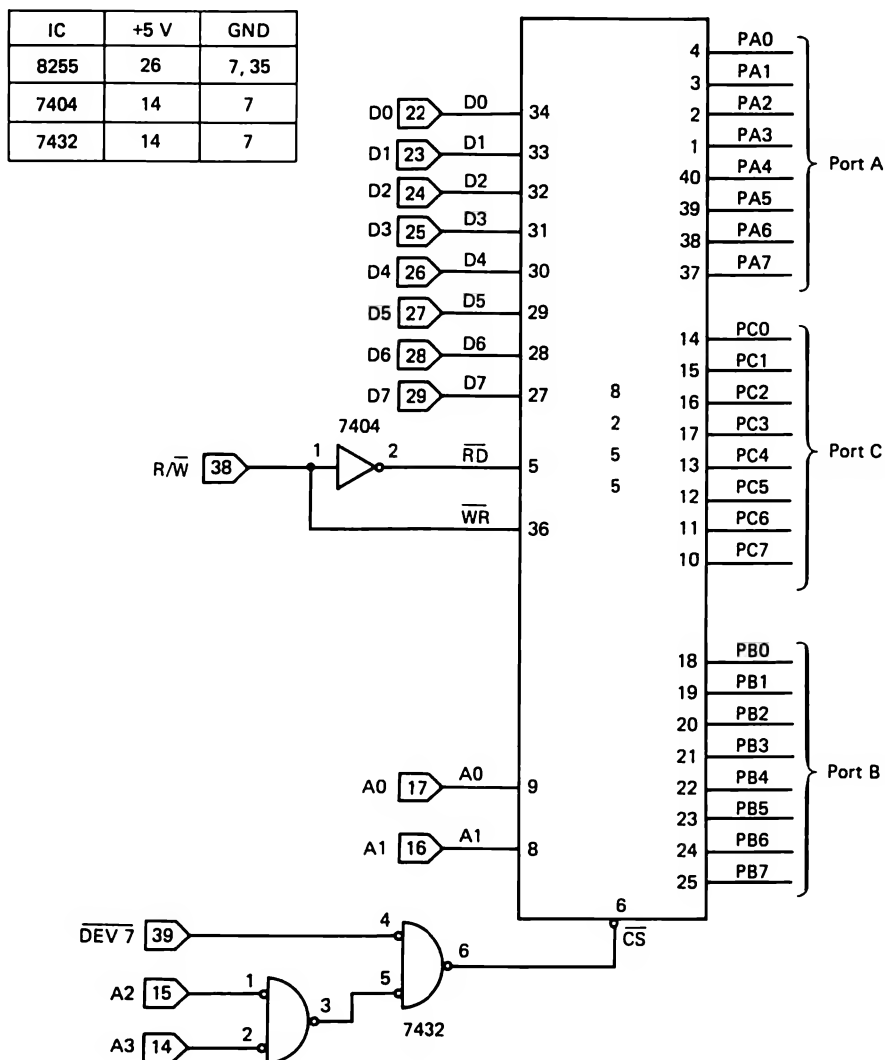


Figure 5-2 Interface circuit between the Apple II and the 8255 PPI. In this circuit full decoding is used and the 8255 occupies the four ports from 49392 to 49395 (assuming that slot 7 is used for the Vector interface card).

memory read signal. The two low-order address lines are connected to the A0 and A1 inputs to allow selection of the four different ports in the 8255 (remember that there are four combinations of 2 bits). Finally, the chip select input is derived from an address decoder of A2, A3, and the device select signal ($\overline{\text{DEV7}}$ in this case).

When the chip select input goes low, the 8255 “looks” at A0 and A1 to determine which port you wish to talk with. It then examines its $\overline{\text{RD}}$ and $\overline{\text{WR}}$ inputs to see which way that data is to flow.

The interface circuit in Fig. 5-2 again takes advantage of the built-in decoding of A4 through A15 within the Apple. Because of this, we need only decode address lines A0 through A3 to provide *full decoding*. Assuming that the Vector plugboard is in slot 7, the device select signal ($\overline{\text{DEV7}}$) will be low for any address between 49392 and 49407. Studying Fig. 5-2, when $\overline{\text{DEV7}}$ is low *AND* A2 *AND* A3 are low, the 8255 will be selected. This corresponds to all addresses between 49392 and 49395.

Example 5-1

List all commands in BASIC that will access the PPI in Fig. 5-2.

Solution There are only two commands: Y=PEEK(X) and POKE X,Y . For the specific addresses involved:

COMMAND	ACTION
Y=PEEK(49392)	Input data from port A
Y=PEEK(49393)	Input data from port B
Y=PEEK(49394)	Input data from port C
Y=PEEK(49395)	Not allowed by the PPI
and	
POKE 49392,Y	Output data Y to port A
POKE 49393,Y	Output data Y to port B
POKE 49394,Y	Output data Y to port C
POKE 49395,Y	Output data Y to the control port

Note that the first four commands cause the $\overline{\text{RD}}$ input to be active and the last four commands cause the $\overline{\text{WR}}$ input to be active.

The commands listed in Example 5-1 actually correspond to a *truth table* for the 8255. This is repeated in Fig. 5-3 in a more general format. Be cautious when using this table, however. Each of the I/O ports (A, B, and C) can be configured as an input or an output port, but not both simultaneously. The specific control word used will determine this. If port A is configured as an output port, you cannot input from this port even though the command Y=PEEK(49392) can be given. The control port must be rewritten with a new control word that causes port A to be an input port.

A1 A0	Operation
0 0	Input data from port A
0 1	Input data from port B
1 0	Input data from port C
1 1	Illegal condition

(a) $\overline{CS} = 0, \overline{RD} = 0$

A1 A0	Operation
0 0	Output data to port A
0 1	Output data to port B
1 0	Output data to port C
1 1	Output data to control group

(b) $\overline{CS} = 0, \overline{WR} = 0$

Figure 5-3 8255 truth table. In (a) the read (or input) operations are summarized. In (b) the write (or output) operations are summarized.

Initializing the PPI from BASIC

Before using the PPI it must be *initialized* for the specific I/O configuration desired. This must be done before any attempt is made to use the circuit in a system.

Initializing the PPI from BASIC is a simple matter consisting of a command of the form **POKE X,Y**, where X is the address of the PPI control port and Y is the desired control word. This word can be determined by referring to Fig. 5-4. This chart details how each bit in the control word is established.

Example 5-2

Determine the control word needed for the following I/O configuration. Port A = output, port B = input, port C (PC0 through PC3) = input, and port C (PC4 through PC7) = output.

Solution Referring to Fig. 5-4:

bit 7 = 1	Defines mode set versus bit set mode
bits 6,5 = 00	Defines mode 0, unconditional I/O
bit 4 = 0	Port A = output
bit 3 = 0	Port C (PC4-PC7) = output
bit 2 = 0	Mode 0
bit 1 = 1	Port B = input
bit 0 = 1	Port C (PC0-PC3) = input

The control word is 10000011 = 131.

Using the hardware shown in Fig. 5-2 and the I/O requirements in Example 5-2, the PPI is initialized with the command

POKE 49395,131

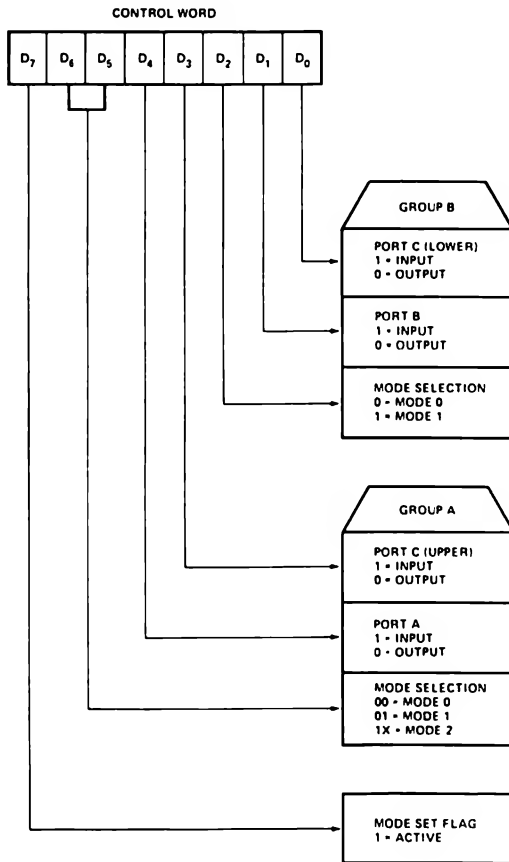


Figure 5-4 Mode definition format for the 8255. The particular control word needed is determined by matching port definitions against those in the chart. Note that for any of the modes (0 through 2) to be active, bit 7 must be a 1. (Courtesy of Intel Corporation.)

PROCEDURE

Step 1. Refer to Fig. 5-5 and carefully wire this circuit on your breadboard. Do not wire the LEDs or flip-flops at this time. Use caution when handling the 8255, as it is an MOS device and subject to damage due to static electricity. Do not handle the IC leads and ground yourself to your breadboard before placing the chip into the socket.

Note. The circuit in Fig. 5-5 uses *partial decoding* to simplify the hardware. Whenever DEV7 is low, the 8255 will be activated and A0 and A1 will determine the specific port selected. This means that the 8255 occupies four *sets* of four ports, with the lowest group at 49392 to 49395 and the highest group at 49404 to 49407.

Step 2. We will test the hardware by wiring ports A and B in parallel and then outputting data to port A and inputting the data from port B. Wire PA0 to PB0, PA1 to PB1, and so on. Refer to Fig. 5-2 for pin numbers.

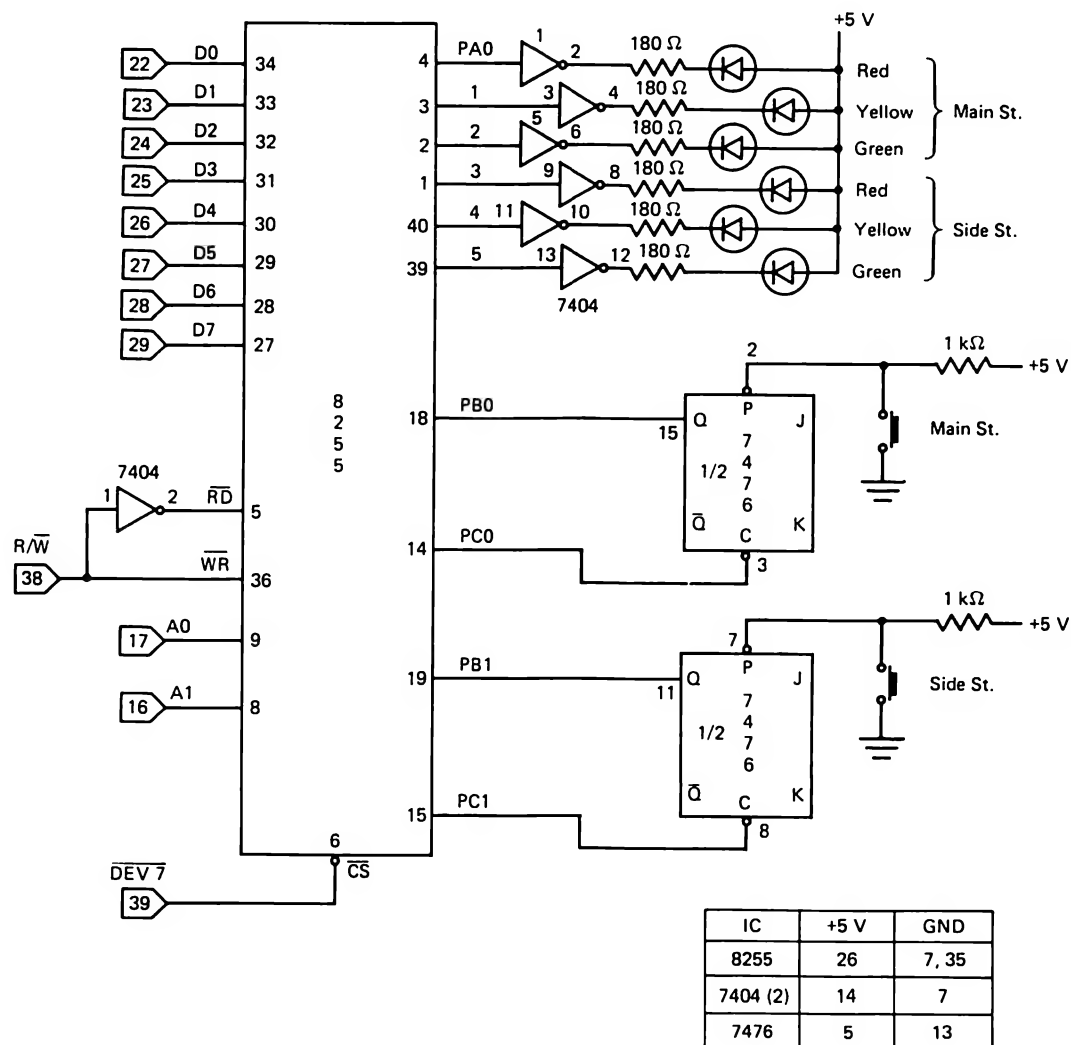


Figure 5-5 Hardware for the Apple traffic light controller interface. Ports A and C are outputs and port B is an input. Cars are detected with two normally open pushbutton switches.

Question 5-1. Write a BASIC program that will test this hardware arrangement. What is the control port address? the control word?

Step 3. To gain more experience with the 8255, you may want to repeat step 2 with ports B and C wired in parallel.

Note. In the next steps we will assemble and test the hardware needed to convert the Apple into a *traffic light controller*. Green, red, and yellow LEDs will simulate the traffic lights and two pushbutton switches will act as car sensors.

Step 4. Remove all wires from ports A, B, and C. Now connect the six LEDs and six 180- Ω resistors to port A using a second 7404 as shown in Fig. 5-5.

Step 5. Test your LED interface by loading and running the following program:

```

10 POKE 49395,130                      :REM MAKE PORT A AN OUTPUT
20 FOR J=0 TO 5
30 POKE 49392,2^J
40 FOR I=1 TO 250: NEXT I
50 NEXT J
60 GOTO 10

```

Question 5-2. What output code word (decimal) is needed to show (a) red on Main St. and green on Side St.? (b) yellow on Main St. and red on Side St.?

Step 6. If your LEDs are working properly, add the car sensors, consisting of one 7476 and two pushbutton switches. Refer to Fig. 5-5 for the details.

Question 5-3. What control word is needed to initialize the 8255 for the traffic light controller circuit in Fig. 5-5?

Note. The 7476 flip-flop will be set by its asynchronous preset input each time a switch is depressed (simulating a car). Because the flip-flop can set only once, the computer must reset this flip-flop so that it can detect the next closure of the switch. In software we can do this by pulsing the PC0 and PC1 lines from high to low and back high again.

Step 7. Test your flip-flop circuit by loading and running the following program. See if you can predict what this program will do.

```

10 HOME
20 POKE 49395,130                      :REM INIT PPI
25 REM NOW RESET THE FLIP-FLOPS
30 POKE 49394,255: POKE 49394,0: POKE 49394,255
40 PRINT "PUSH THE MAIN ST PUSHBUTTON"
50 GOSUB 500: IF B(0)=1 THEN POKE 49392,7: GOTO 70
60 GOTO 50
70 PRINT "PUSH THE SIDE ST PUSHBUTTON"
80 GOSUB 500: IF B(1)=1 THEN POKE 49392,56: GOTO 100
90 GOTO 80
100 INPUT "PUSH RETURN TO RESET THE FLIP-FLOPS ";A$

```

```

120 POKE 49392,0
130 GOTO 10
500 Y=PEEK(49393)
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN

```

Note. As discussed in Experiment 4, it is necessary to convert the data from the input port (49393) to binary to test the individual bits. This is done using the method developed in Experiment 4 and appears in the program as the subroutine in lines 500 through 560. The array B(J) holds the binary value of the variable Y.

Step 8. When your hardware works successfully through step 7, you are ready to write the traffic light controller program. The following requirements should be met:

1. Main St. is the priority street. It must have a minimum of 30 s of green before changing to red. In addition, it should only change to red if 30 s has elapsed and at least one car is waiting on Main St.
2. The green light on Side St. should stay green until either 30 s has elapsed or three or more cars are waiting on Main St.
3. In all cases, lights should sequence from green, to yellow (for 4 s), to red.

Note. A solution to this step is provided at the end of this experiment.

Step 9. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the “Discussion” and “Procedure” sections.

SOLUTIONS TO QUESTIONS

- 5-1. The control port is 49395 (using the lowest set of addresses). Because port C is undefined, there are four possible control words that configure ports A and B as desired. They are: 130, 131, 138, and 139. One possible test program is

```

10 HOME
20 POKE 49395,130 :REM INIT PPI
30 INPUT "DATA TO OUTPUT TO PORT A IS ";A
40 POKE 49392,A :REM 49392 IS PORT A
50 INPUT "PUSH RETURN TO SEE THE DATA INPUT FROM PORT B ";B$
60 PRINT PEEK(49393) :REM 49393 IS PORT B
70 IF PEEK(49393)=A THEN PRINT "PORT TEST IS OK": GOTO 90

```

```

80 PRINT "PORT TEST FAILS"
90 INPUT "AGAIN ";B$
100 IF B$="YES" THEN 30
110 END

```

Another test program that lets the Apple do all the testing is

```

1000 HOME
1010 POKE 49395,130           :REM INIT PPI
1020 FOR J=0 TO 255
1030 POKE 49392,J
1040 IF PEEK(49393)=J THEN 1060
1050 PRINT "TEST FAILS ON OUTPUT ";J
1060 NEXT J
1070 PRINT "TEST OVER"
1080 END

```

5-2. (a) 33; (b) 10

5-3. Port A = output to LEDs. Port B = input from car sensors. Port C (lower) = output to reset car sensors. Control word = 130 or 138.

Step 8 (solution). A possible solution is shown below (refer also to the flowchart in Fig. 5-6).

```

10 HOME
20 20 POKE 49395,130           :REM INIT PPI
25                               :REM RESET SENSORS
30 POKE 49394,255: POKE 49394,0: POKE 49394,255
40 POKE 49392,12              :REM MAIN IS GREEN SIDE IS RED
50 D=30: GOSUB 600            :REM WAIT 30 S
60 GOSUB 500: IF B(1)<>1 THEN 60 :REM CAR ON SIDE ST?
65                               :REM RESET SIDE ST SENSOR
70 POKE 49394,3: POKE 49394,1: POKE 49394,3
80 POKE 49392,10              :REM MAIN IS YELLOW SIDE IS RED
90 D=4: GOSUB 600            :REM WAIT 4 S
100 POKE 49392,33             :REM SIDE IS GREEN MAIN IS RED
110 C=0                       :REM CAR COUNTER
120 FOR K=1 TO 30             :REM 30 LOOPS AT 1 S EACH
130 GOSUB 500: IF B(0)=1 THEN 160 :REM CAR ON MAIN?
140 D=1: GOSUB 600            :REM DO LOOPS AT 1 S EACH
150 GOTO 200
155                               :REM RESET MAIN ST SENSOR
160 POKE 49394,3: POKE 49394,2: POKE 49394,3
170 C=C+1                    :REM ADD 1 TO CAR COUNTER
180 IF C<>3 THEN 200          :REM 3 CARS?
190 K=30
200 NEXT K
210 POKE 49392,17             :REM SIDE IS YELLOW MAIN IS RED
220 D=4: GOSUB 600            :REM WAIT 4 S
230 GOTO 40

```

```

500 Y=PEEK(49393)
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN
600 FOR T=1 TO D*750
610 NEXT T
620 RETURN

```

:REM SAMPLE SENSORS

```

:REM TIME DELAY ADJUST FOR 1 S
PER VALUE OF D

```

Light code

(12)

(10)

(33)

(17)

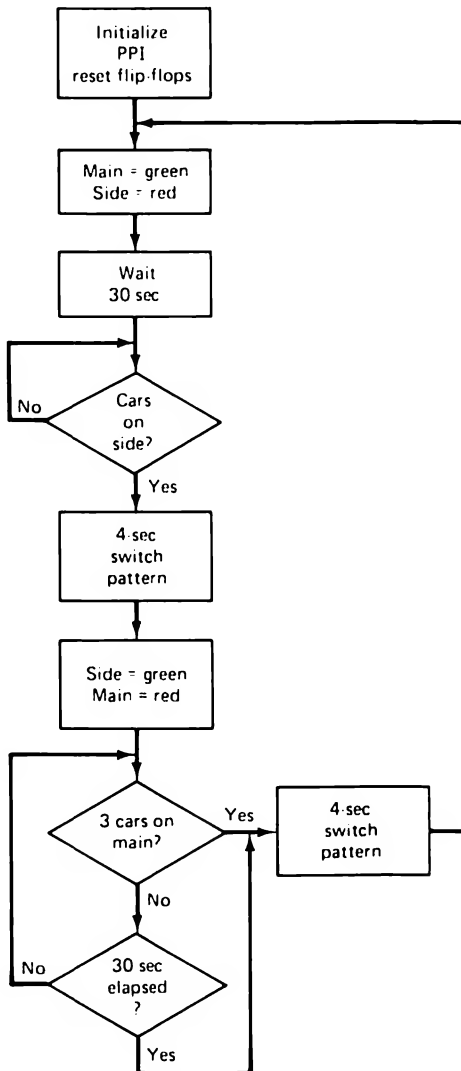


Figure 5-6 Traffic light controller program flowchart.

Note. This program should work with your hardware, but if your switches have excessive bounce time, erratic operation may occur. This is because the computer will reset the flip-flop only once. If the switch bounce time is too long, it will set again. The cure is a brief time delay in lines 65 and 160:

FOR I = 1 to 5: NEXT I

This allows the switch time to stop bouncing before the computer resets the flip-flop.

EXPERIMENT 6

INTERFACING THE 6820 PROGRAMMABLE INTERFACE ADAPTER

OVERVIEW

In this experiment you will wire a *6820 programmable interface adapter* (PIA) to the Apple. You will learn how to program this device for basic input and output by repeating the traffic light controller problem from Experiment 5.

OBJECTIVES

The key points to be learned from this experiment are:

1. The 6820 is a standard programmable I/O device intended for use in 6800/6500 family microcomputer systems.
2. This chip features 16 I/O lines that can be individually programmed as input or output pins.
3. Like the 8255 PPI chip, the 6820 occupies four memory locations. Two of these ports are used for control of the device and the remaining two for data input/output.

PARTS LIST

- 1 6820 programmable interface adapter (Jameco MC6820 or MC6821)
- 1 7404 hex inverter
- 1 7476 dual JK flip-flop
- 2 red LEDs
- 2 green LEDs

- 2 yellow LEDs
- 6 180- Ω resistors (brown-gray-brown)
- 2 1-k Ω resistors (brown-black-red)
- 2 pushbutton normally open (PBNO) switches

Note. Throughout this experiment the PIA referred to is the MC6820. However, the MC6821 is an identical part and may be substituted for the MC6820 throughout this experiment.

DISCUSSION: ANOTHER PROGRAMMABLE I/O DEVICE

The 6820 programmable interface adapter (PIA) is another example of the trend in integrated electronics toward *programmable* interface devices. Such devices can free the microcomputer of the tedious task of continually monitoring its I/O ports. Also, because they are programmable, these intelligent I/O adapters can be controlled via software and actually change their hardware appearance to suit the device being interfaced. For example, a parallel printer may require as many as nine output lines for data and strobe signals and three to four input lines for printer status signals. But a burglar alarm system may require 15 input lines and only a single output line for an alarm condition. The 6820 can accommodate both of these interfacing problems with a simple change of data written to its control register.

If you have read through Experiment 5, you have probably realized that the 6820 is basically another 8255. The 8255 was designed by Intel Corp. for use with its 8080 and 8085 family of microprocessors. But as we have seen, it can easily be adapted to the 6502 microcomputer (that is, the Apple). The 6820 was originally designed by Motorola for use with the 6800 microprocessor. It is also available from MOS Technology as the 6520 and considered a support device for 6500 series microprocessors (MOS Technology also offers the MCS6522, an enhanced version of the 6520/6820). Because of the similarity between the 6800 and 6502, we will see that the PIA is easily interfaced to the Apple.

Figure 6-1 is a block diagram of the 6820 illustrating the bidirectional data bus interface and 16 programmable I/O pins. Four *handshaking* lines are also provided, called CA1, CA2, CB1, and CB2. Handshaking I/O was introduced in Experiment 5 and is covered in detail for the 8255 in Experiment 11.

Although the microcomputer normally only inputs and outputs data from the PIA, it is possible to cause the PIA to *interrupt* the microcomputer from its present task and have it immediately service an I/O device. This is done by causing the PIA to generate an interrupt pulse. The 6820 provides two such signals, labeled $\overline{\text{IRQA}}$ (interrupt request A) and $\overline{\text{IRQB}}$ (interrupt request B). More detail on this type of I/O can again be found in Experiment 11.

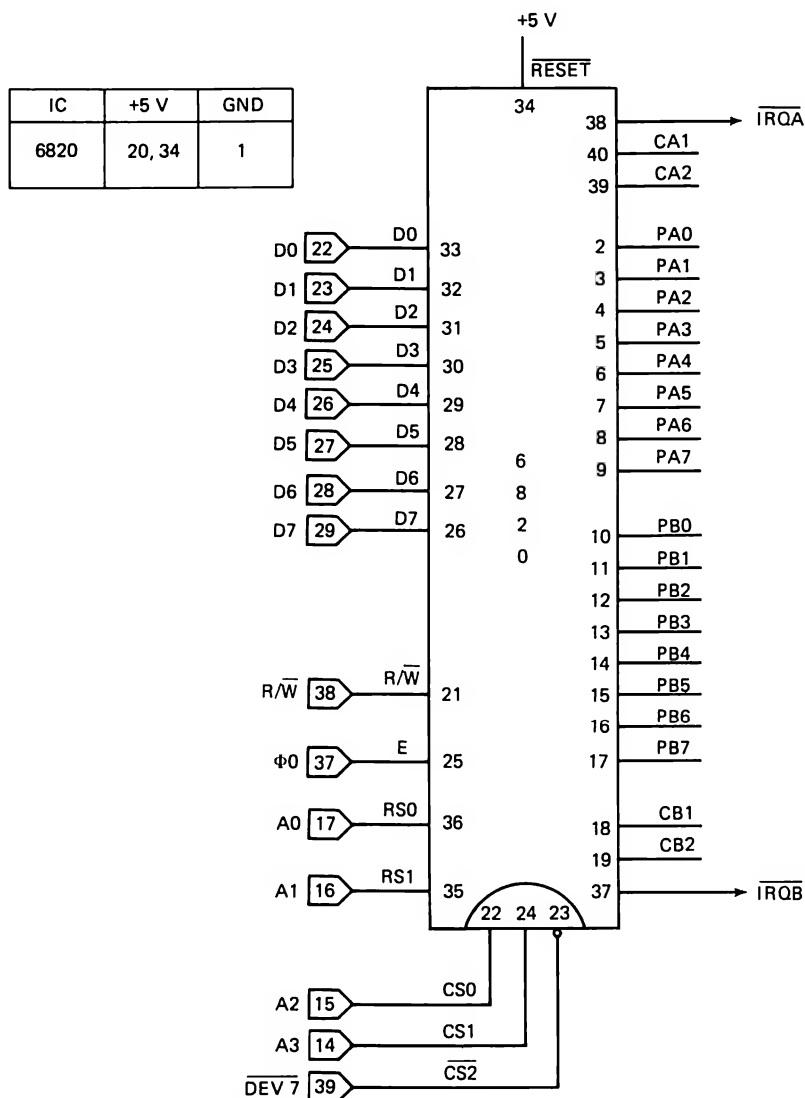


Figure 6-2 6820 PIA interface to the Apple. The circuit occupies ports 49404 through 49407 when the Vector card is in slot 7.

2. Three chip select inputs are provided, which in this case eliminates the need for address decoder gates (refer to Fig. 5-2 for a contrast). When CS0 AND CS1 are high, AND CS2 is low, the 6820 will be enabled.
3. Once enabled, address lines A0 and A1, connected to the RS0 and RS1 register select inputs, determine which of four (actually six) internal

registers will be selected. This means the 6820 will appear as four memory locations to the microcomputer system.

4. The four internal registers may be read from or written to by the microcomputer. The $\overline{R/\overline{W}}$ line tells the 6820 which way data is to go. When $\overline{R/\overline{W}}$ is high, a read operation is indicated. When $\overline{R/\overline{W}}$ is low, a write operation is indicated.

Example 6-1

Indicate the range of addresses occupied by the 6820 PIA interface circuit shown in Fig. 6-2.

Solution Because the $\overline{CS2}$ chip select input is connected to the Apple's $\overline{DEV7}$ device select signal, the address must be in the range 49392 to 49407 (refer to Experiment 2 if this is not clear to you). Now to enable the 6820, $CS0$ and $CS1$ must both be high and $\overline{CS2}$ low. This means that the offset to add to the base address (49392) is 12 ($A3 A2 A1 A0 = 11XX = 12$). The four combinations of $A0$ and $A1$ result in addresses in the range 49404 to 49407, all enabling the circuit.

The four combinations of $A0$ and $A1$ should define four registers within the PIA. However, by being a bit crafty, they actually define six. Table 6-1 helps explain. The definition of each port is set by separate control registers labeled CRA (control register A) and CRB (control register B). When $RS1$ and $RS0 = 00$ or 10 , bit 2 of the corresponding control register defines whether the *data direction register* (DDR) or *data buffer* is accessed. In this way six locations are defined with only 2 bits. The control register should be written to before accessing the data direction register or data buffer since the value of control register bit 2 is generally unknown.

The two data direction registers specify the port pins that will be inputs or outputs. A logic 1 specifies an output, a 0 an input.

Example 6-2

Write a BASIC program to configure port A of the 6820 interface in Fig. 6-2 with bits 0, 1, 5, and 7 as inputs and bits 2, 3, 4, and 6 as outputs.

TABLE 6-1

	RS1	RS0	Address	Function
CRA bit 2=0	0	0	49404	Port A data direction register
CRA bit 2=1	0	0	49404	Port A data buffer
CRA bit 2=X	0	1	49405	Port A control register
CRB bit 2=0	1	0	49406	Port B data direction register
CRB bit 2=1	1	0	49406	Port B data buffer
CRB bit 2=X	1	1	49407	Port B control register

TABLE 6-2 PIA CONTROL REGISTER FORMAT*

Determine Active CA1 (CB1) Transition for Setting Interrupt Flag IRQA(B) 1—(bit b7)

- b1 = 0 : IRQA(B) 1 set by high-to-low transition on CA1 (CB1).
- b1 = 1 : IRQA(B) 1 set by low-to-high transition on CA1 (CB1).

IRQA(B) 1 Interrupt Flag (bit b7)

Goes high on active transition of CA1 (CB1); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

CA1 (CB1) Interrupt Request Enable/Disable

- b0 = 0 : Disables IRQA(B) MPU Interrupt by CA1 (CB1) active transition.¹
- b0 = 1 : Enable IRQA(B) MPU Interrupt by CA1 (CB1) active transition.

1. IRQA(B) will occur on next (MPU generated) positive transition of b0 if CA1 (CB1) active transition occurred while interrupt was disabled.

b7	b6	b5	b4	b3	b2	b1	b0
IRQA(B) 1 Flag	IRQA(B) 2 Flag	CA2 (CB2) Control			DDR Access	CA1 (CB1) Control	

IRQA(B) 2 Interrupt Flag (bit b6)

CA2 (CB2) Established as Input (b5 = 0): Goes high on active transition of CA2 (CB2); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.
CA2 (CB2) Established as Output (b5 = 1): IRQA(B) 2 = 0, not affected by CA2 (CB2) transitions.

Determines Whether Data Direction Register Or Output Register is Addressed

- b2 = 0 : Data Direction Register selected.
- b2 = 1 : Output Register selected.

CA2 (CB2) Established as Output by b5 = 1

b5 b4 b3
1 0 0 → CA2,
(Note that operation of CA2 and CB2 output functions are not identical)

b3 = 0 : Read Strobe With CA1 Restore

CA2 goes low on first high-to-low E transition following an MPU Read of Output Register A; returned high by next active CA1 transition.

b3 = 1 : Read Strobe with E Restore

CA2 goes low on first high-to-low E transition following an MPU Read of Output Register A; returned high by next high-to-low E transition.

→ CB2

b3 = 0 : Write Strobe With CB1 Restore

CB2 goes on low on first low-to-high E transition following an MPU Write into Output Register B; returned high by the next active CB1 transition.

b3 = 1 : Write Strobe With E Restore

CB2 goes low on first low-to-high E transition following an MPU Write into Output Register B; returned high by the next low-to-high E transition.

→ Set/Reset CA2 (CB2)

CA2 (CB2) goes low as MPU writes b3 = 0 into Control Register.

CA2 (CB2) goes high as MPU writes b3 = 1 into Control Register.

CA2 (CB2) Established as Input by b5 = 0

b5 b4 b3
0 0 0 →

CA2 (CB2) Interrupt Request Enable/
Disable

b3 = 0 : Disables IRQA(B) MPU Interrupt by CA2 (CB2) active transition.¹
b3 = 1 : Enables IRQA(B) MPU Interrupt by CA2 (CB2) active transition.

1. IRQA(B) will occur on next (MPU generated) positive transition of b3 if CA2 (CB2) active transition occurred while interrupt was disabled.

Determines Active CA2 (CB2) Transition for Setting Interrupt Flag IRQA(B) 2— (bit b6)

b4 = 0 : IRQA(B) 2 set by high-to-low transition on CA2 (CB2).

b4 = 1 : IRQA(B) 2 set by low-to-high transition on CA2 (CB2).

Solution The proper control word to write to DDRA is

01011100 = 92

The program would be

```

10 POKE 49405,0      :REM BIT 2 = 0 DDRA ACCESS
20 POKE 49404,92     :REM DEFINE I/O PINS
30 POKE 49405,4      :REM BIT 2 = 1 DATA BUFFER ACCESS
40 END

```

Comparing the 6820 and 8255

When programming the 8255 in Experiment 5, we spoke of various programming *modes*. There was mode 0 for unconditional I/O, mode 1 for handshaking I/O, and mode 2 for bidirectional I/O. Is the 6820 also programmed by selecting a particular operating mode?

The answer is no—although it will support the 8255's modes 0 and 1. This programming is accomplished through the control register associated with port A (CRA) or port B (CRB).

Because handshaking capabilities are handled by the CA and CB control pins, there is no need to select a mode 0 or mode 1. If you need handshaking capabilities, then use the CA and CB control pins; if not, then ignore these pins.

Programming of the control register is quite complex, allowing great flexibility in the use of the CA and CB control pins as **BUSY/READY** and **ACKNOWLEDGE** signals. Interrupt-driven I/O can also be accommodated by the 6820 via the **IRQA** and **IRQB** outputs, as discussed previously. The function of these pins is again set by the control registers. Table 6-2 summarizes the control register format for the 6820. Because handshaking I/O is discussed in detail in Experiment 11, we will not go into detail about programming the 6820 control registers beyond basic I/O requirements.

It is possible to use the CA2 and CB2 control pins in a simple bit set/reset mode. This is illustrated in Tables 6-2 and 6-3. Similarly, it is possible

TABLE 6-3

CRA input bits 5 4 3			Output CA2
1	1	0	0
1	1	1	1
CRB input bits 5 4 3			Output CB2
1	1	0	0
1	1	1	1

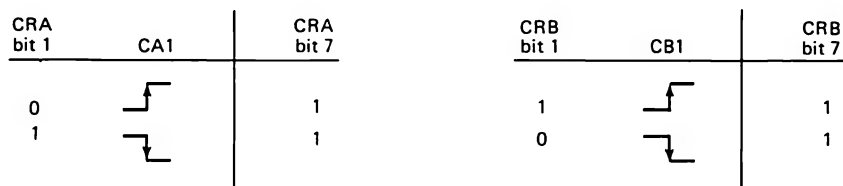


Figure 6-3 The CA1 and CB1 control lines will cause bit 7 of the corresponding control register to be set when a rising or falling edge occurs, as shown in the figure.

to monitor the CA1 and CB1 input control pins. This is shown in Fig. 6-3 and in Table 6-2. In this case, bit 7 of the control register will be set when the desired edge (either rising or falling) occurs as specified by control register bit 1. Bit 7 will be reset by a subsequent read operation of the corresponding data register.

Example 6-3

Using the hardware interface of Fig. 6-2, write a BASIC program to detect when the CA1 input goes high and output a logic 0 to the CA2 and CB2 output pins.

Solution The initial bit pattern for CRA is

7 6 5 4 3 2 1 0 ← bit

X X 1 1 1 X 1 X (32+16+8+2)

set bit 7 when CA1 = 

set CA2=1

Similarly for CRB:

7 6 5 4 3 2 1 0 ← bit

X X 1 1 1 X X X (32+16+8)

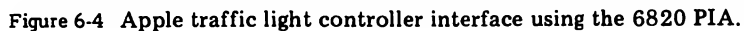
set CB2=1

The program initializes the control registers and then waits for bit 7 of CRA to go high. When this occurs the CA2 and CB2 control lines are reset.

```

10 POKE 49405,4           :REM ACCESS PORT A DATA BUFFER
20 Y= PEEK (49404)         :REM MAKE SURE CRA7=0
30 POKE 49405, (32+16+8+2) :REM INIT CRA
40 POKE 49407, (32+16+8)   :REM INIT CRB
50 IF PEEK(49405)<128 THEN 50 :REM WAIT FOR RISING EDGE
60 POKE 49407, (32+16+2)   :REM MAKE CB2 LOW
70 POKE 49405, (32+16)     :REM MAKE CA2 LOW
80 END

```



PROCEDURE

Step 1. Refer to Fig. 6-4 and wire this circuit on your breadboard. Do not wire the LEDs or two flip-flops at this time.

Step 2. Following the test procedure developed in Experiment 5, wire ports A and B in parallel.

Question 6-1. Write a BASIC program to test this circuit. Have the computer test all combinations of both ports acting as inputs and outputs.

Step 3. If the hardware passes the test program in Question 6-1, test the CA and CB control lines by devising a test circuit to go along with the BASIC program given in Example 6-3. A solution is provided at the end of this experiment.

Step 4. Study the circuit in Fig. 6-4 and compare it to the 8255 interface in Fig. 5-5. These two circuits are nearly identical except that the 8255 version uses three ports, whereas the 6820 can use only two.

Step 5. If you are solving the traffic light controller problem for the first time, turn back to steps 4 through 7 of Experiment 5 and perform these steps using the interface circuit in Fig. 6-4. Remember to change addresses to match the 6820 hardware configuration.

Step 6. With the LEDs and flip-flops working properly, write the traffic light controller program as described in step 8 of Experiment 5. A solution is provided at the end of this experiment.

Step 7. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Procedure" and "Discussion" sections.

Note. Experiments 7, 8, 9, 11, and 13 are all written using the 8255 PPI. If you prefer, all of these experiments except number 11 can be done using the 6820 PIA. The port addresses will have to be changed and the initialization commands will be different.

 SOLUTIONS TO QUESTION

6-1. One possible solution is

```

10 HOME
20 A=0 :REM DDRA CODE=INPUT
30 B=255 :REM DDRB CODE=OUTPUT
40 GOSUB 500 :REM PROGRAM THE PORTS
50 PRINT "TESTING A AS INPUT, B AS OUTPUT"
60 PRINT
```

```

70  FOR J=0 TO 255
80  POKE 49406,J
90  IF PEEK(49404)=J THEN 120
100 PRINT "ERROR AT ";J;" READ ";PEEK(49404)
110 PRINT
120 NEXT J
130 A=255                :REM DDRA CODE=OUTPUT
140 B=0                  :REM DDRB CODE=INPUT
150 GOSUB 500             :REM PROGRAM THE PORTS
160 PRINT "TESTING B AS INPUT, A AS OUTPUT"
170 PRINT
180 FOR J=0 TO 255
190 POKE 49904,J
200 IF PEEK(49406)=J THEN 230
210 PRINT "ERROR AT ";J;" READ ";PEEK(49406)
220 PRINT
230 NEXT J
240 PRINT "TEST COMPLETE"
250 END
500 REM SUBROUTINE TO PROGRAM PORTS A AND B
510 POKE 49405,0          :REM DDRA ACCESS
520 POKE 49407,0          :REM DDRB ACCESS
530 POKE 49404,A          :REM PROGRAM PORT A
540 POKE 49406,B          :REM PROGRAM PORT B
550 POKE 49405,4          :REM A=DATA BUFFER
560 POKE 49407,4          :REM B=DATA BUFFER
570 RETURN

```

Step 3 (solution). Figure 6-5 shows the circuit. When the program is run, both LEDs should come on (CA2 and CB2 = 1). Depressing the switch will cause both LEDs to go off.

Step 8 (solution)

```

10  HOME
15  POKE 49405,0: POKE 49407,0          :REM INIT PIA
17  REM NOW PROGRAM A=INPUT, PB0,1=OUTPUT PB6,7=INPUT
20  POKE 49406,3: POKE 49404,255: POKE 49407,4: POKE 49405,4
25                                     :REM RESET SENSORS
30  POKE 49406,3: POKE 49406,0: POKE 49406,3
40  POKE 49404,12                    :REM MAIN IS GREEN SIDE IS RED
50  D=30: GOSUB 600                  :REM WAIT 30S
60  GOSUB 500: IF B(7)<>1 THEN 60      :REM CAR ON SIDE ST?
65                                     :REM RESET SIDE ST SENSOR
70  POKE 49406,3: POKE 49406,1: POKE 49406,3
80  POKE 49404,10                    :REM MAIN IS YELLOW SIDE IS RED
90  D=4: GOSUB 600                  :REM WAIT 4 S
100 POKE 49404,33                    :REM SIDE IS GREEN MAIN IS RED

```

```

110 C=0                                :REM CAR COUNTER
120 FOR K=1 TO 30                      :REM 30 LOOPS AT 1 S EACH
130 GOSUB 500: IF B(6)=1 THEN 160      :REM CAR ON MAIN?
140 D=1: GOSUB 600                     :REM DO LOOPS AT 1 S EACH
150 GOTO 200
155                                    :REM RESET MAIN ST SENSOR
160 POKE 49406,3: POKE 49406,2: POKE 49406,3
170 C=C+1                              :REM ADD 1 TO CAR COUNTER
180 IF C<>3 THEN 200                   :REM 3 CARS?
190 K=30
200 NEXT K
210 POKE 49404,17                      :REM SIDE IS YELLOW MAIN IS RED
220 D=4: GOSUB 600                     :REM WAIT 4 S
230 GOTO 40
500 Y=PEEK(49406)                      :REM SAMPLE SENSORS
510 FOR J=7 TO 0 STEP -1
520 B(J)=1

```

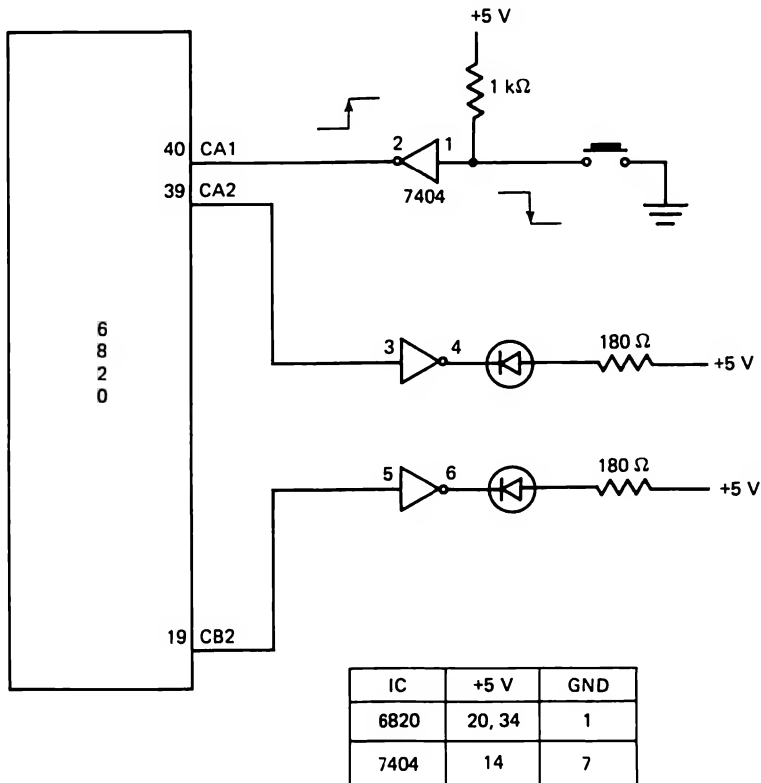


Figure 6-5 Test circuit for step 3 and Example 6-3. Pressing the switch should turn OFF the LEDs.

```
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN
600 FOR T=1 TO D*750                :REM TIME DELAY ADJUST FOR 1S
                                      PER VALUE OF D
610 NEXT T
620 RETURN
```

Note. This program should work with your hardware, but if your switches have excessive bounce time, erratic operation may occur. This is because the computer will reset the flip-flop only once. If the switch bounce time is too long, it will set again. The cure is a brief time delay in lines 65 and 160:

FOR I = 1 to 5: NEXT I

This allows the switch time to stop bouncing before the computer resets the flip-flop.

3

Special Interfacing Problems

The experiments in this section should not be attempted until the concepts in Part 2 have been mastered. This section covers special problems that occur when interfacing the Apple to the outside world.

EXPERIMENT 7

— HARDWARE INTERFACING TECHNIQUES, PART 1: INPUTS —

OVERVIEW

In this experiment you will interface several different input sensors to the Apple. These include a *magnetic switch*, a *temperature sensor*, and a *photo-cell*. Input techniques using the *game I/O connector* will be explained and control programs using BASIC will be developed.

OBJECTIVES

The key points to be learned from this experiment are:

1. The outside world generally presents data to the computer in non-TTL-compatible voltage levels.
2. The Apple can only interpret TTL levels, and others may damage the computer.
3. An analog comparator circuit is commonly used to convert nonstandard voltages to TTL.
4. One 8-bit input port allows the computer to monitor eight different sensors.

PARTS LIST

- 1 7404 hex inverter
- 1 8255 programmable peripheral interface (Jameco DP8255)
- 1 LM339 quad comparator (Radio Shack 276-1712)
- 1 LM334 temperature sensor (Radio Shack 276-1734)

- 1 cadmium sulfide photoresistor (Radio Shack 276-116)
- 1 magnetic switch (Radio Shack 49-495)
- 3 1-k Ω resistors (brown-black-red)
- 2 10-k Ω resistors (brown-black-orange)
- 1 220- Ω resistor (red-red-brown)
- 2 10-k Ω variable resistors (pots)

DISCUSSION: INPUTTING DATA FROM THE OUTSIDE WORLD

The true power of the microcomputer becomes clear when we begin interfacing it to the “*outside world*.” Imagine typing on your keyboard and being able to turn on or off any light or appliance within your home, or having your computer sense the temperature in your living room and turn on the furnace or air conditioner as appropriate. The possibilities are nearly endless. The main obstacle in doing any of these things is the electronics required between the computer and the controlled device.

Figure 7-1 illustrates the problem. The microcomputer is a binary machine communicating only in 1’s and 0’s. Unfortunately, most of the information we wish to monitor is not in this digital format. Temperature, for example. The temperature could be 10°F or 80°F or any value in between. Yet the computer expects a simple yes or no type of input. Quantities such as temperature, pressure, humidity, and velocity are all *analog* in nature and cannot simply be described as on or off. These types of inputs can be interfaced to the computer, but they require a special *analog-to-digital* converter circuit. This technique will be examined in detail in Experiment 10.

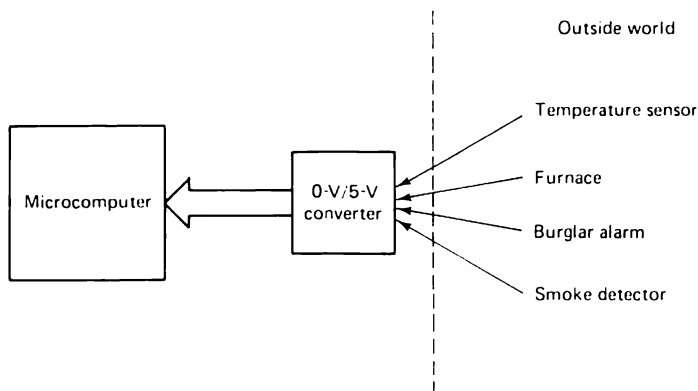


Figure 7-1 The outside world is interfaced to the microcomputer through a circuit that must convert the sensor’s information to 0-V and 5-V levels compatible with the computer’s electronics.

In many cases we do not need to know the exact value of the analog information. In this case we can force the input to be of a yes or no nature. For example, is the temperature 60°F, or is the humidity less than 50%, or is the furnace off? All of these questions can be answered yes or no. The problem now becomes one of converting the sensor's ON/OFF status to 0-V and 5-V logic levels that the computer can interpret. This is the 0-V/5-V converter box in Fig. 7-1.

The next few sections illustrate some examples of the circuitry required to convert the ON/OFF information of the sensor to standard logic levels.

Magnetic Switch Interface

One of the most common and simplest devices to interface to a computer is a mechanical switch. This device certainly meets our criterion of having only an ON or OFF condition. In this example let's see how the open and closed switch contacts are converted to standard logic levels.

Magnetic switches are commonly used in burglar alarm systems to detect door or window openings. Two types of switches are possible. In one, the switch contacts *close* when the magnet engages. In the other, the switch contacts *open* when engaged. Figure 7-2 is an example of the former type. The switch itself is housed in the plastic package shown on the right. The left-most package is simply the magnet. One advantage to this type of switch is that the contacts are not exposed to the environment. This allows reliable contact closures even under dirty conditions.

Figure 7-3 illustrates an interface to the Apple using the 8255 PPI chip. The magnetic switches shown are being used to detect door openings and the contacts close when the magnets are engaged. With the door closed, the 7404 inverter input is held at ground (0 V) and the 8255 inputs (PC0 and PC1) are high. An open door disengages the magnets, opening the contacts. The in-



Figure 7-2 Typical magnetic switch. When the two units are brought within 1 inch of each other, the switch contacts will close. The unit shown is a Radio Shack 49-495.

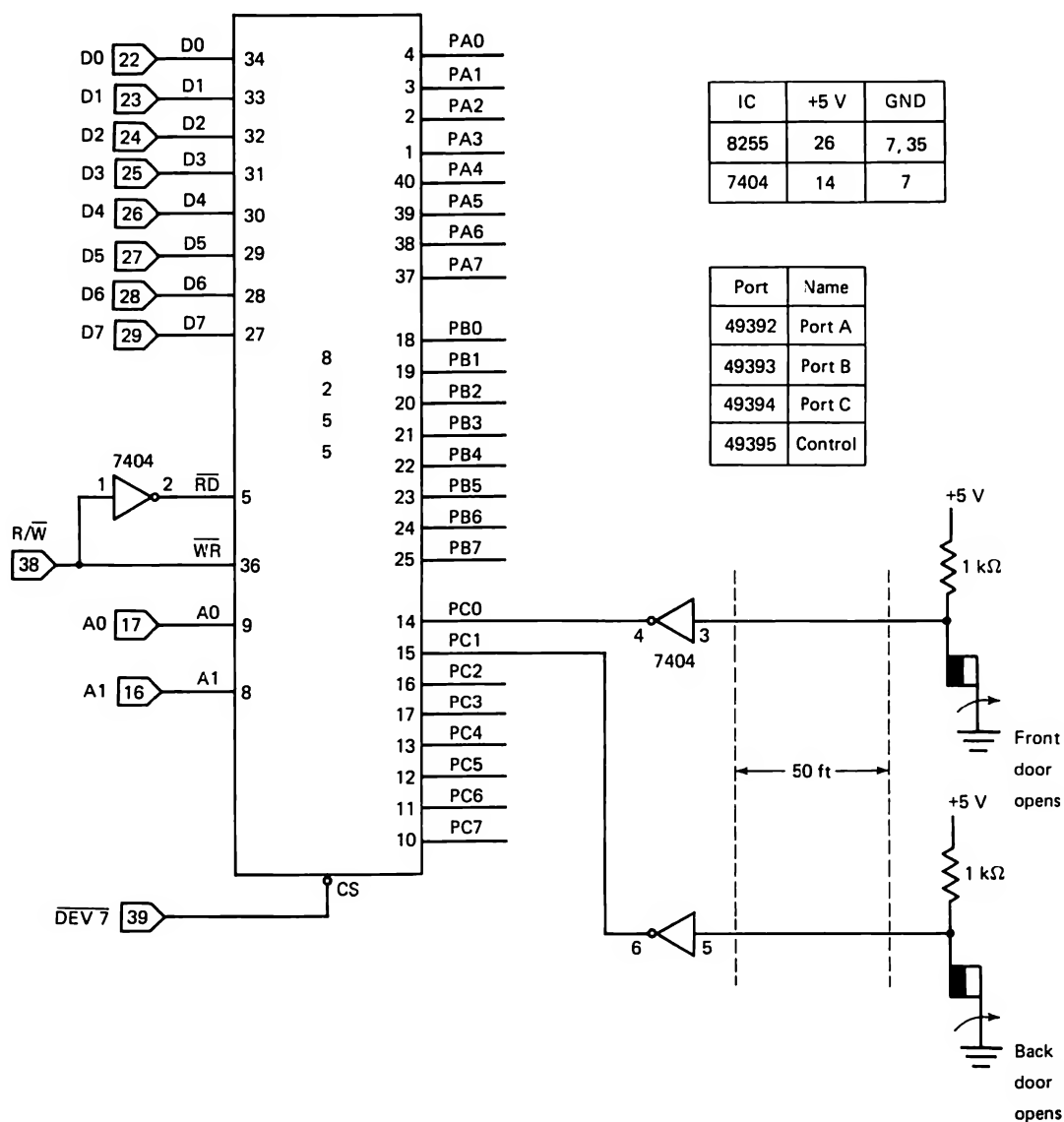


Figure 7-3 Two magnet switches are used to monitor the front and back doors of a home. When either door opens, the 7404 inverter input goes high and the input port receives a low logic level.

verter input now goes high and the 8255 input goes low. A BASIC program to check the two doors might be:

```

10 HOME
20 POKE 49395,137                :REM INIT PPI. A,B=OUTPUTS, C=INPUT
30 GOSUB 500: IF B(0)<>1 THEN PRINT "FRONT DOOR OPEN"
40 IF B(1)<>1 THEN PRINT "BACK DOOR OPEN"
50 END
500 Y=PEEK(49394)                :REM SAMPLE SENSORS
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN

```

Note that the now standard subroutine (lines 500 through 560) is again required to convert the input data to binary. This technique was first discussed in Experiment 4.

Using one 8255 chip, 24 inputs can be monitored. The wiring from each switch to the computer will require a pair of conductors, but the 7404 and pull-up resistor may be located at the computer site.

Sensing Light and Dark

In this example and the next we will see how the analog nature of a sensor can be converted to a digital format.

There are numerous instances where it would be desirable for your computer to sense the relative amount of light in a room or out-of-doors. For example, an outdoor light sensor could detect nightfall and the computer could then activate a yard light. A light sensor might also be used as part of a burglar alarm system or to count objects proceeding down an assembly line.

A useful integrated circuit for accomplishing these types of interfaces is the LM339 *analog quad comparator* shown in Fig. 7-4. This chip has four identical comparators in one 14-pin package. Each comparator has two inputs, labeled (+) and (-). Electronically, the comparator examines the voltages on its two inputs. If the (+) input is largest, the output will switch to an *open-circuit* condition. An external pull-up resistor is used to pull the output to +5 V for this case. If the (-) input is the largest, the output switches to 0 V. A particularly nice feature of the LM339 is that only a few microvolts of difference between the two inputs is required for the output to switch states. Typical switching times are less than 1 μ s.

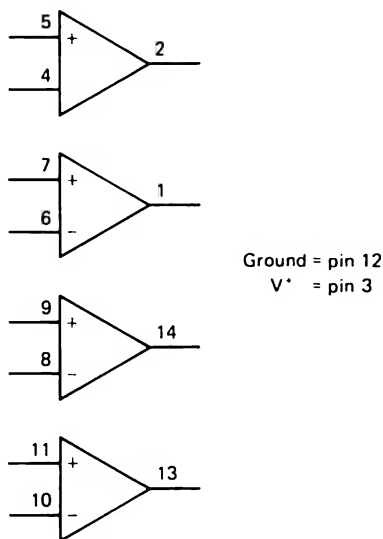


Figure 7-4 LM339 quad comparator. Four comparators are included in one standard 14-pin dual-in-line package (DIP). The output switches between an open circuit when the (+) input is greater than the (-) input, and 0 V when the (-) input is greater than the (+) input.

The LM339 is also useful for converting voltages which exceed normal TTL values to standard levels. This is because the comparator inputs are rated to withstand voltages as high as +36 V (the minimum is -0.3 V).

A simple circuit for detecting the presence of light and converting this information to 0-V and 5-V levels is shown in Fig. 7-5. This circuit uses a *photoresistor* (shown in Fig. 7-6) and one-fourth of the LM339 quad comparator. When the comparator (+) input (pin 7) has a more positive voltage than the (-) input (pin 6), the output of the comparator goes OFF and is

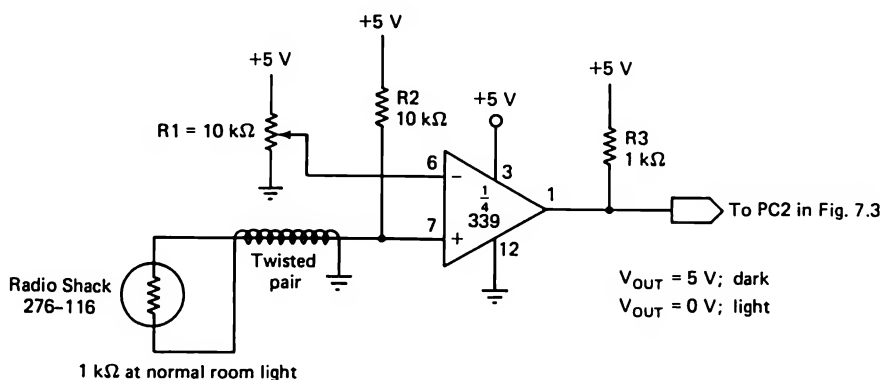


Figure 7-5 Light-sensing circuit with adjustable threshold. This circuit uses one of the four comparators in the LM339 package. The photoresistor has a resistance which is low ($\sim 100 \Omega$) in the presence of light but large ($\sim 1 \text{ M}\Omega$) in darkness. By adjusting R1, the specific light-dark switching threshold can be set.

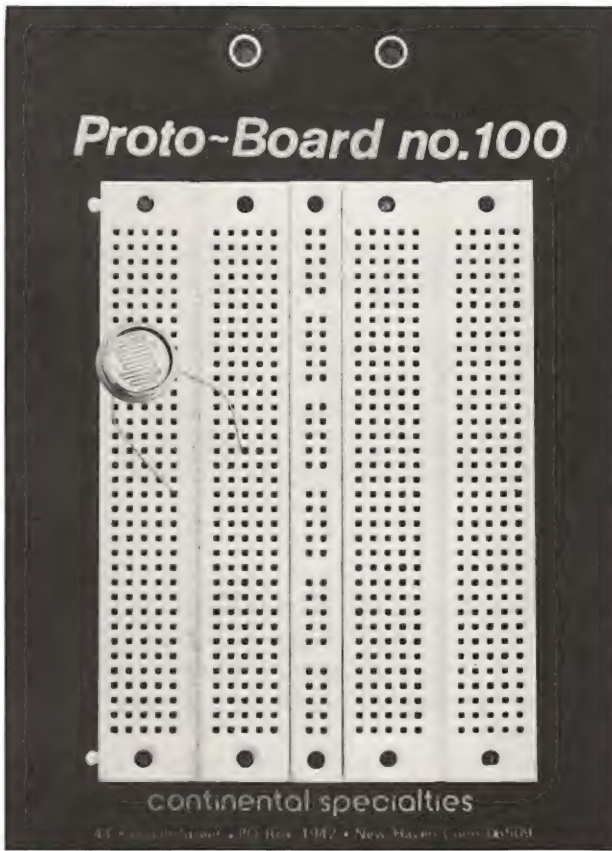


Figure 7-6 Typical light sensor. The unit shown is a cadmium sulfide photoresistor (Radio Shack 276-116).

pulled to 5 V by R3. When the (-) input is most positive, the output is 0 V. Because the photoresistor's resistance varies from hundreds of ohms in bright light to megohms in darkness, the voltage at pin 7 will vary from 0 V (bright) to 5 V (dark).

The circuit can be adjusted by setting the light conditions to the switching point and then adjusting R1 so that the comparator output (pin 1) just switches to the desired value (5 V for dark or 0 V for light). A BASIC test program for aligning the interface might be (assuming that PC2 in Fig. 7-3 is the input bit)

```

10 HOME
20 POKE 49395,137           :REM INIT PPI
30 VTAB(12): HTAB(18)
40 GOSUB 500: IF B(2)=1 THEN PRINT "DARK ": GOTO 30
50 PRINT "LIGHT"
60 GOTO 30

```

```

500 Y=PEEK(49394)           :REM SAMPLE THE SENSOR
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF B(J)-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN

```

Again a two-conductor cable can be used to connect the photoresistor to the interface electronics. With four comparators in one package, four such sensors can be monitored.

One application of the photoresistor interface that should not be overlooked is a *touch sensor* input to the computer. When an input is desired, the user simply covers the photoresistor with his or her finger. Because the sensor can be mounted behind a plate of glass, this technique is useful when it is desired to limit actual access to the computer for security reasons.

Sensing Temperature

Temperature is another analog quantity that can be monitored by the Apple. In this instance we will not be concerned with the specific temperature, but rather in the ability to determine if the temperature is above or below some threshold value. This is consistent with our approach (in this experiment) to convert analog information to digital.

Figure 7-7 illustrates the electrical interface. An LM334 *temperature sensor* is used to provide a current that is proportional to temperature. With the component values shown, the voltage applied to the comparator (+) input will increase 10 mV/°K. For example, with $V^+ = 5$ V, the output voltage across R2 is typically 3.1 V, but will rise or fall as the temperature changes. By adjusting R3 to a particular value, we may detect when the temperature exceeds (or falls below) this limit.

Example 7-1

Refer to Fig. 7-7. Assume that the room temperature is 20°C (68°F) and $V_{R2} = 3.1$ V. Adjusting R3, what voltage should be applied to pin 4 of the comparator if we wish to detect temperatures above 32°C (90°F)?

Solution The change in temperature is $32^\circ\text{C} - 20^\circ\text{C} = 12^\circ\text{C}$. Because a Kelvin degree is the same as a Celsius degree, V_{R2} will increase by $12^\circ \times 10 \text{ mV}/^\circ\text{K} = 120 \text{ mV}$. R3 should be set so that pin 4 is $3.1 \text{ V} + 0.12 \text{ V} = 3.22 \text{ V}$.

The LM334 temperature sensor is contained in a TO-92 plastic package and looks identical to a small plastic transistor. The 220- Ω resistor (R1) can be soldered directly to the two device leads and the remaining two connections (V^+ and V^-) are brought to the interface electronics through a two-conductor cable. Because the device is a *current source*, the resistance of the cable, and therefore its length, is not critical.

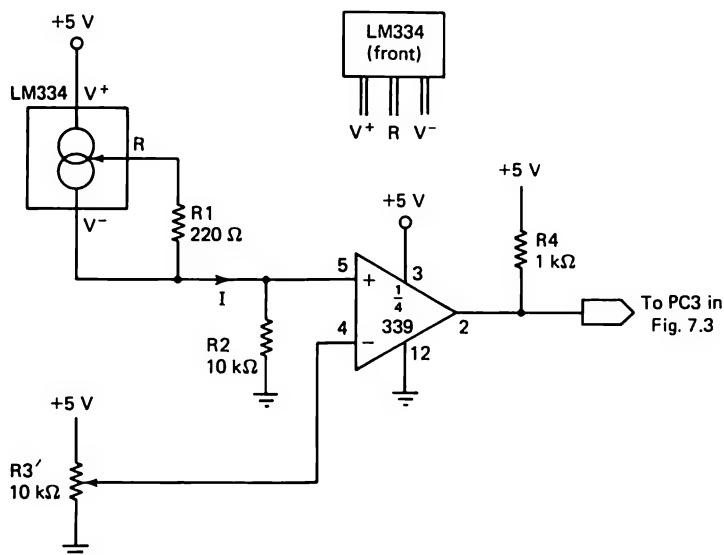


Figure 7-7 Sensing temperature with the Apple. The LM334 output current increases with temperature such that V_{R2} will rise $10 \text{ mV}/^\circ\text{K}$. When the LM339 comparator (+) input (pin 5) exceeds the threshold voltage established by R3 on pin 4, its output pulls high through R4. For temperatures below this point, the output is low.

The best way to calibrate the circuit in Fig. 7-7 is to place the LM334 at the temperature condition you wish to detect and then measure the output voltage across R2. Now adjust R3 until the voltage at pin 4 just equals this value. If you do not have a voltmeter, the following program can be run. With the LM334 at the switching temperature, slowly adjust R3 until the screen displays "HOT." This program assumes that the LM339 output is connected to PC3 in Fig. 7-3.

```

10 HOME
20 POKE 49395,137 :REM INIT PPI
30 VTAB(12):HTAB(18)
40 GOSUB 500: IF B(3)=1 THEN PRINT "HOT ": GOTO 30
50 PRINT "COLD"
60 GOTO 30
500 Y=PEEK(49394) :REM SAMPLE TEMP SENSOR
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN

```

Smoke Detector Interface

Occasionally, the signal we wish to monitor is not steady but of a *pulsing* nature: for example, the alarm signal on a digital clock or smoke detector. In this case, the interface circuitry must *latch* the alarm condition to prevent the computer from assuming multiple alarms. Generally, this also requires some means of resetting the latch once the alarm has ceased.

In this example let us see how to interface a smoke detector to the Apple. Figure 7-8 illustrates a common type. This is the Radio Shack *ionization smoke alarm* and it produces a burst of 3-kHz audio at 4 to 6 Hz when activated.

The interfacing scheme consists of simply paralleling wires with the smoke detector's internal audio alarm. In the case of the Radio Shack unit, this involves prying the front cover off (voiding the warranty) and connecting two wires to the outermost connections of the audio alarm.

Because the alarm is pulsing, the interface circuit must use a *flip-flop* to hold the alarm input to the computer at a steady (nonpulsing) logic level. The interface circuit is illustrated in Fig. 7-9. When the smoke alarm is activated, the pulsing alarm signal is applied as a clock signal to the 7476 JK flip-flop. With J wired high and K low, the Q output will switch high on the first pulse from the alarm. This can be monitored by the computer for further action.

Because the flip-flop output will remain high even after the alarm disappears, the clear input must be pulsed low to reset the circuit. This is accomplished by pulsing the PA0 line.



Figure 7-8 Smoke detector and alarm. Illustrated is the Radio Shack ionization smoke alarm 49-454. The two wires shown are in parallel with the internal speaker.

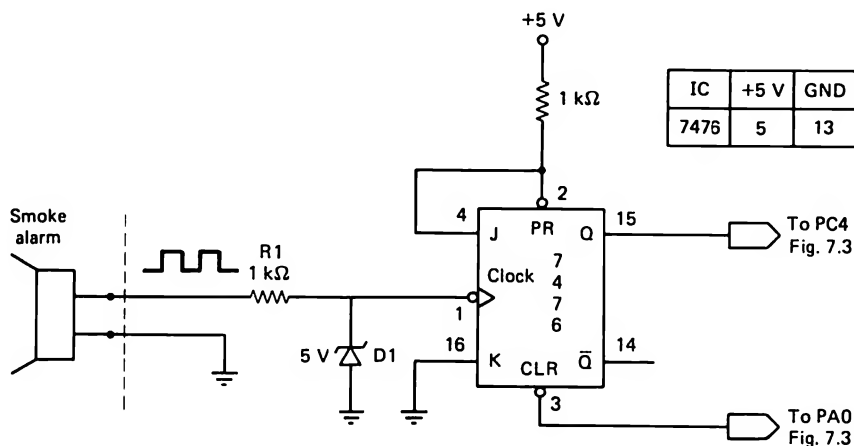


Figure 7-9 Apple smoke detector interface. PC4 and PA0 refer to the 8255 schematic diagram in Fig. 7-3. The flip-flop is necessary because the alarm signal is pulsing. D1 and R1 protect the input from large or negative input voltages.

The following program will clear the flip-flop (in case it powers up already set) and then wait for an alarm condition.

```

10 HOME
20 POKE 49395,137                                :REM INIT PPI
30 POKE 49392,0: POKE 49392,1                     :REM RESET FLIP-FLOP
40 GOSUB 500: IF B(4)<>1 THEN 40
50 VTAB(10):HTAB(18): PRINT "ALARM"
60 END
500 Y=PEEK(49394)                                :REM CHECK FLIP-FLOP
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN

```

Using the Game Connector for Input

The Apple *game connector* is a special IC socket located in the upper right corner of the motherboard. This socket is commonly used for *game paddles* or *joysticks*. However, it can also be used for inputting information to the microcomputer just as we have been discussing in this experiment.

There are two restrictions you should be aware of when using this connector.

1. The game connector has only 3 bits or channels (IN0 through IN2).
2. Each channel is accessed as a *separate memory location*; you do not test individual bits.

Figure 7-10 illustrates the pinning of this socket. The three input lines are available on pins 2 through 4. Notice that 5 V and ground are also provided. The four output port lines will be discussed in Experiment 8.

Each input line is TTL compatible and its logic level can be determined by PEEKing the address shown. If you refer to the memory map in Fig. 2-3, you will see that the three port addresses are within the block of 128 locations referred to as *built-in I/O*. When PEEKing one of these three locations, a result greater than 127 indicates a 1, less than 128 indicates a 0.

As an example of how to use this connector, Fig. 7-11 illustrates a magnetic door switch interface similar to the one discussed in Fig. 7-3. The software required to monitor this switch is very simple.

```

10 HOME
20 IF PEEK(49249) < 128 THEN PRINT "FRONT DOOR OPEN"
30 IF PEEK(49250) < 128 THEN PRINT "BACK DOOR OPEN"
40 END

```

This program is similar to the one presented earlier in the section on the magnetic switch interface. In fact, it is simpler because we do not have to convert the input data to binary and test individual bits. There is also no 8255 initialization. Of course, there are only three input lines compared to 24 with the 8255. Nevertheless, when you have a need to monitor only a few TTL inputs, the game I/O socket may be a good choice.

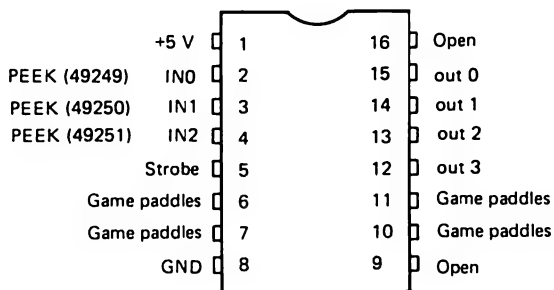


Figure 7-10 Pin numbers and descriptions for the Apple game I/O connector. Three input lines are available.

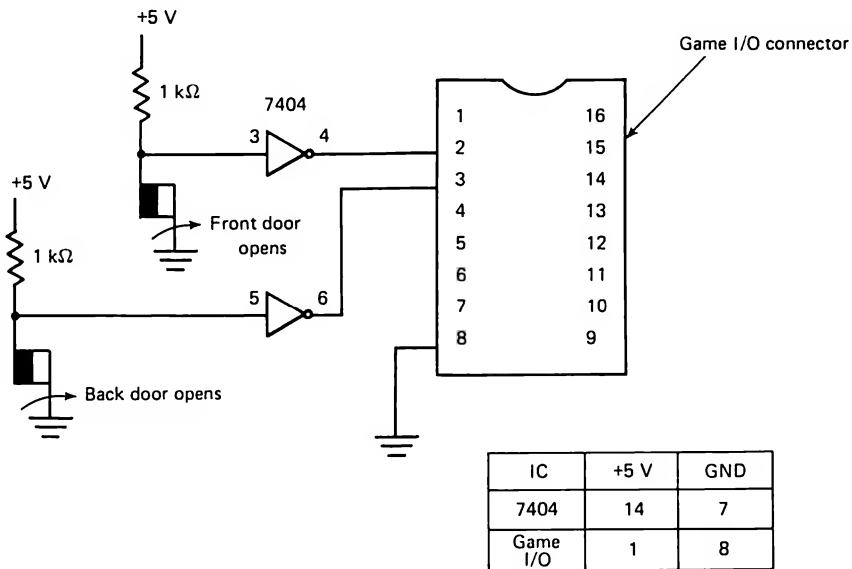


Figure 7-11 Magnetic door switch interface using the game I/O connector.

PROCEDURE

Step 1. Study the circuit in Fig. 7-3 and wire the 8255 PPI and the RD/WR 7404 on your breadboard. For the moment connect nothing else to the 8255.

Question 7-1. We can test that the hardware is functional by programming all three PPI ports to be inputs and selectively grounding one input at a time while holding all others high. What are the eight different input combinations that BASIC will see as each pin is grounded?

Question 7-2. What command should be used to initialize the PPI?

Step 2. Load and run the following program, testing all three ports by touching one input pin at a time to ground with the other seven connected to +5 V. Do not proceed until all ports check out.

```

10 HOME
20 POKE 49395,155 :REM PROGRAM ALL PORTS AS INPUTS
30 INPUT "WHICH PORT DO YOU WISH TO TEST (A, B OR C) ";P$
40 IF P$="A" THEN N=49392: GOTO 70
50 IF P$="B" THEN N=49393: GOTO 70
60 N=49394
70 VTAB(10):HTAB(18): PRINT PEEK(N)
80 GOTO 70

```

Step 3. Connect two wires to the contacts on one magnetic switch. Connect one of these wires to ground and the other wire to pin 3 of the 7404 using a 1-k Ω pull-up resistor as shown in Fig. 7-3. Connect the 7404 output (pin 4) to the PC0 input (pin 14) of the 8255.

Step 4. Using the program given in the discussion as a guide, write a similar program to detect when the magnets are close together ("closed") or far apart ("open"). Don't forget to initialize the 8255.

Step 5. Now interface the light-sensing circuit shown in Fig. 7-5. Connect the LM339 output (pin 1) to PC2 (pin 16) of the 8255. Load and run the test program given in the "Discussion." Adjust the setting of R1 until you can get the computer to display "LIGHT" and "DARK" as the room lights are turned on and off.

Step 6. Using another comparator in the LM339, construct the temperature-sensing circuit shown in Fig. 7-7. Connect the comparator output (pin 2) to PC3 (pin 17) of the 8255. Now load and run the program given in the "Discussion."

Step 7. Adjust R3 until the screen just displays "COLD." Now pinch the LM334 between your fingers. The screen should display "HOT" in 5 to 10 s.

Note. All three sensors should now be interfaced and adjusted. In the following step we will develop a control program to monitor particular conditions of all three sensors.

Step 8. Write a BASIC program that:

1. Displays the status of all three sensors on the screen.
2. Asks for the alarm condition that should cause the computer to display or flash "ALARM." For example, if the door is open, the temperature is hot and the light is dark, sound the alarm.

One possible solution is provided at the end of this experiment.

Step 9. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

Note. Do not disassemble the hardware for this experiment, as portions of it will be used in Experiment 8.

SOLUTIONS TO QUESTIONS

- 7-1. When the least significant bit is grounded, the input port will see 11111110 or 254₁₀. The seven other combinations are: 253, 251, 247, 239, 223, 191, and 127.

7-2. POKE 49395,155

Step 8 (solution)

```

10 HOME: POKE 49395,137 :REM INIT PPI
20 PRINT "WHAT IS THE ALARM CONDITION?"
30 PRINT
40 INPUT "DOOR OPEN OR CLOSED ";D$
50 IF D$="OPEN" THEN D=0: GOTO 80
60 IF D$="CLOSED" THEN D=1: GOTO 80
70 GOTO 40
80 INPUT "LIGHT CONDITION: BRIGHT OR DARK ";L$
90 IF L$="BRIGHT" THEN L=0: GOTO 120
100 IF L$="DARK" THEN L=1: GOTO 120
110 GOTO 80
120 INPUT "TEMPERATURE CONDITION: HOT OR COLD ";T$
130 IF T$="COLD" THEN T=0: GOTO 160
140 IF T$="HOT" THEN T=1: GOTO 160
150 GOTO 120
160 HOME
170 D$="OPEN "
180 GOSUB 500: IF B(0)=1 THEN D$="CLOSED"
190 L$="BRIGHT"
200 IF B(2)=1 THEN L$="DARK "
210 T$="COLD"
220 IF B(3)=1 THEN T$="HOT "
230 VTAB(6): HTAB(1): PRINT "THE DOOR IS: ";D$
240 VTAB(8): PRINT "THE LIGHT IS: ";L$
250 VTAB(10): PRINT "THE TEMPERATURE IS: ";T$
260 VTAB(14): HTAB(18)
270 IF B(0)=D AND B(2)=L AND B(3)=T THEN PRINT "A L A R M"
280 PRINT " ": GOTO 170
500 Y=PEEK(49394)
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN

```

Note. If you wish to simulate this program without building the hardware, replace line 500 with

```
500 Y=INT(1000*RND(1)): IF Y>255 THEN 500
```

EXPERIMENT 8

HARDWARE INTERFACING TECHNIQUES, PART 2: OUTPUTS

OVERVIEW

In this experiment you will wire an *npn* transistor as a mechanical relay driver. A solid-state relay will also be interfaced and driven directly with TTL. Finally, an ultrasonic interface will be constructed for controlling 120-V AC devices. Control programs using BASIC will be developed.

OBJECTIVES

The key points to be learned from this experiment are:

1. Few real-world devices can be driven directly by a microcomputer.
2. A *mechanical relay* can be used to control most electronic and electro-mechanical devices. It can be driven by a microcomputer via an output port and external driver transistor.
3. *Solid-state* relays can be driven directly with TTL but may not handle the current and voltage of their mechanical counterparts.
4. When isolation between the controlled device and the computer is necessary, *opto-couplers* and *ultrasonic* techniques can be used.

PARTS LIST

- 1 7400 quad NAND gate
- 1 7404 hex inverter
- 1 8255 programmable peripheral interface (Jameco DP8255)

- 1 LM386 integrated amplifier (Radio Shack 276-1731)
- 1 LM567 tone decoder (Radio Shack 276-1721)
- 1 TIL111 or 4N25 opto-coupler (Radio Shack 276-132)
- 1 magnetic switch (Radio Shack 49-495)
- 1 piezoelectric buzzer (Radio Shack 273-060)
- 1 SPDT DIP relay (Radio Shack 275-240)
- 1 general-purpose diode
- 1 general-purpose *npn* transistor
- 2 40-kHz transducers (The Micromint Inc., 561 Willow Avenue, Cedarhurst, NY 11516)
- 1 LED
- 1 0.05- μ F capacitor
- 1 0.1- μ F capacitor
- 1 0.033- μ F capacitor
- 1 0.005- μ F capacitor
- 1 0.0047- μ F capacitor
- 1 10- μ F capacitor
- 1 100- μ F capacitor
- 2 10-k Ω potentiometers (pots)
- 1 100- Ω resistor (brown-black-brown)
- 1 150- Ω resistor (brown-green-brown)
- 1 220- Ω resistor (red-red-brown)
- 1 330- Ω resistor (orange-orange-brown)
- 2 1-k Ω resistors (brown-black-red)

DISCUSSION: THE MICROCOMPUTER AS A CONTROLLER

BASIC programming presents new challenges when your computer can turn on the light across the room or sound an audio alarm when two switch contacts are opened. In Experiment 7 we developed techniques for getting information into the Apple. In this experiment we want to finish that story by studying output port techniques.

The output devices your computer can control may be analog or digital in nature. For example, to control a DC motor the computer could output a binary number. When converted to a voltage level, the magnitude of this number will determine the speed of the motor. An interface of this type will require a special *digital-to-analog* converter circuit. This type of interface will be discussed in detail in Experiment 9.

In this experiment we will interface several output devices that the computer can control with simple ON and OFF commands. These include

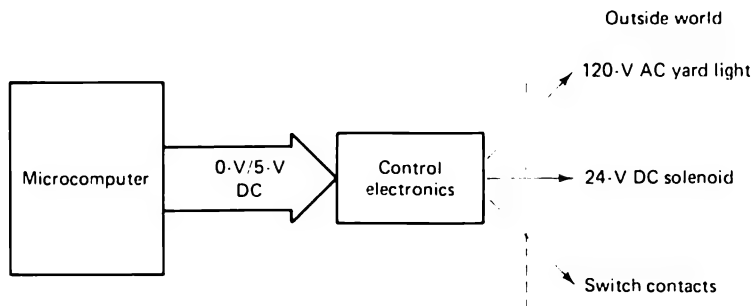


Figure 8-1 A microcomputer can be used to control any number and type of “outside world” devices provided that the proper control electronics is provided between the computer and the outside world.

lights, relays, home appliances, sprinklers, and most any other device whose control function can be switched ON and OFF. Once again the main obstacle to accomplishing these interfaces is the control electronics required between the computer and outside world. This is illustrated in Fig. 8-1.

The control electronics are necessary because the typical microcomputer is a 5-V DC machine, but the typical outside world device is anything but 5-V DC! For example, it might be a 24-V DC solenoid on a sprinkler system, a pair of switching contacts on a slide projector, or a 120-V AC yard light.

In addition to their non-TTL compatibility, these outside world devices can also present a *hazard* to your microcomputer (if you have ever seen a digital circuit board that “accidentally” encountered 120-V AC, you know what I mean!). Therefore, for some interfaces, another requirement for the control electronics is *isolation* (no direct electrical connection) between the computer and control circuitry.

Mechanical Relays

Mechanical relays have been used for many years to allow low-voltage DC control circuits to control high-voltage and current AC or DC devices. The basic technique is illustrated in Fig. 8-2a. When S1 is closed, current is allowed to flow through the relay coil to ground, causing it to become an *electromagnet*. This, in turn, pulls in the moving metal arm of the relay, causing the lamp to light as the AC current path through the normally open (NO) contact and arm is completed. Releasing the switch turns off the light.

Note that the control circuit consists of S1, the 5-V DC source, and the relay coil. Depending on the relay type, the control current required is usually quite small. On the other hand, the controlled circuit consists of the relay contacts, the light, and the 120-V AC source. The current capabilities of this circuit are limited by the size of the relay contacts. Typical values for a small 5-V relay are 1 to 3 A. Larger relays may handle correspondingly

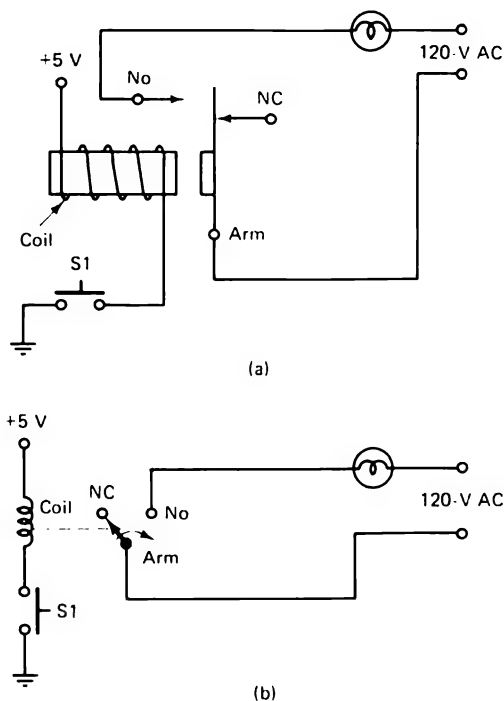


Figure 8-2 Typical relay control circuit. The circuit in (a) presents a pictorial view of the relay. The symbol in (b) is more common on schematic diagrams. In either case, depressing S1 closes the relay contacts and activates the AC light.

higher currents. The schematic symbol for the relay is often separated into a coil and contact set linked by a dashed line, as shown in Fig. 8-2b.

Figure 8-3 illustrates an interface between the Apple and a small 5-V DC relay. The relay is being used to control an LED powered by a 12-V DC source. This is an example of how non-TTL voltages can be interfaced to the microcomputer. An 8255 programmable peripheral interface chip is used and port A is programmed as an output port. Transistor Q1 takes the place of the pushbutton switch in Fig. 8-2. When PA0 goes high, this voltage will turn on the transistor, causing a short circuit to exist between its collector (C) and emitter (E) terminals. This is analogous to closing S1 in Fig. 8-2. Current can now flow through the relay coil and the switch contacts, lighting the LED.

You might wonder why the relay could not be connected directly to the 8255 output port (programming PA0 to be a 0 could then turn on the relay). The problem with this is that the relay coil current, although not large, is still too large to be handled by the 8255 alone or most any TTL circuit.

For the relay shown in Fig. 8-3, the manufacturer indicates a coil resistance of $56\ \Omega$. This means that the relay will draw about 90 mA ($5\text{-V}/56\ \Omega$) of current when energized. The 8255 can sink 1.6 mA of current maximum, and most TTL gates are limited to 16 mA maximum. The transistor, however, can easily handle this current.

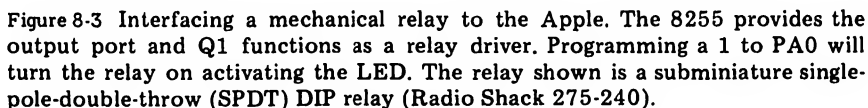


Figure 8-3 Interfacing a mechanical relay to the Apple. The 8255 provides the output port and Q1 functions as a relay driver. Programming a 1 to PA0 will turn the relay on activating the LED. The relay shown is a subminiature single-pole-double-throw (SPDT) DIP relay (Radio Shack 275-240).

When PA0 switches low, the transistor will turn off and become an open circuit between its collector and emitter terminals. This will turn the relay off. It will also generate a momentary, but large, inductive “kickback” voltage at the collector terminal. This is similar to the spark produced when the breaker points on your car’s distributor open and close. It is due to current being interrupted while flowing through an inductor or coil. Diode D2 protects the transistor from damage due to this voltage transient by shorting the voltage to ground. Resistors R1 and R2 protect the transistor and LED junctions from excessive current.

Solid-State Relays

The main advantages of using a mechanical relay are its simplicity and high current and high voltage handling capabilities. However, being mechanical, it does present certain problems. The switch contacts are subject to wear and pitting as they continually open and close. And because the contacts are exposed to the environment, they may eventually become contaminated with dirt. The switching time of a relay is also relatively slow.

A *solid-state* relay has no moving parts or contacts to pit and wear out. It can switch from ON to OFF in the time it takes to turn on a transistor (microseconds).

One example of a solid-state relay and interface circuit is shown in Fig 8-4. The key component in this circuit is the TIL111 *opto-coupler*. This is a six-pin integrated circuit consisting of an infrared LED (IRED) and photo infrared-detecting transistor. When the computer outputs a 1 to PA1 of the 8255, the 7404 inverter grounds the IRED cathode, causing it to emit light. This, in turn, saturates the phototransistor, causing it to become a short circuit between its collector (C) and emitter (E): in effect, shorting the “relay” contacts. In this example the piezoelectric buzzer then emits a loud 4.8-kHz tone suitable as a small alarm signal.

One disadvantage to the circuit in Fig. 8-4 is that the relay contacts, pins 4 and 5 of the opto-coupler, can only handle direct current. Figure 8-5 illustrates a slight modification to this circuit that allows it to control AC as well as DC.

The four diodes in this circuit are connected in a *bridge* configuration and they force the current to flow through the phototransistor in one direction only (DC). In this example the relay contacts are inserted in series with the low-voltage AC control wires to the solenoid on a gas furnace. When the 8255 outputs a 1, the IRED and phototransistor turn on, providing a low-resistance path through the diodes and transistor, turning on the furnace.

Control with BASIC is very simple.

```

10 POKE 49395,137      :REM INIT PPI, A AND B OUTPUTS, C AN INPUT
20 POKE 49392,2         :REM TURN ON FURNACE
30 END
```

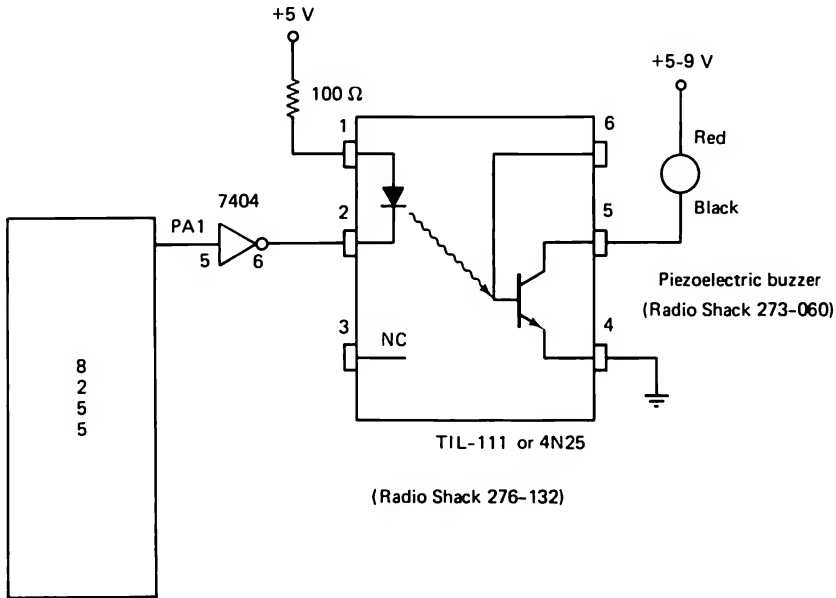


Figure 8-4 A solid-state relay can be built using an opto-coupler. When activated by the computer, pins 4 and 5 of the opto-coupler become a short circuit turning on the piezoelectric buzzer. Refer to Fig. 8-3 for the 8255 pin numbers.

Commercially available solid-state relays with current capabilities as high as 50 A are available. They usually contain an opto-isolated *TRIAC* molded into a plastic package. These are particularly simple to use, as only the input and output terminals are accessible.

Ultrasonic Interface

An important advantage of using an opto-coupler is the *isolation* that it provides between the control and controlled circuits. There is no electrical connection between the two because the light path is used to pass the ON/OFF control signal. This is important because it protects the control circuit (and computer!) from faults that may occur in the usually higher voltage controlled circuit.

Another type of circuit that provides even greater isolation than the opto-coupler is the ultrasonic interface illustrated in block diagram form in Fig. 8-6. A 40-kHz (ultrasonic) sound wave is transmitted under computer control through the air. When the receiver detects this signal it turns on a relay, activating the controlled device. Depending on the transmitter and receiver circuits, distances of 20 to 30 ft between the two are possible.

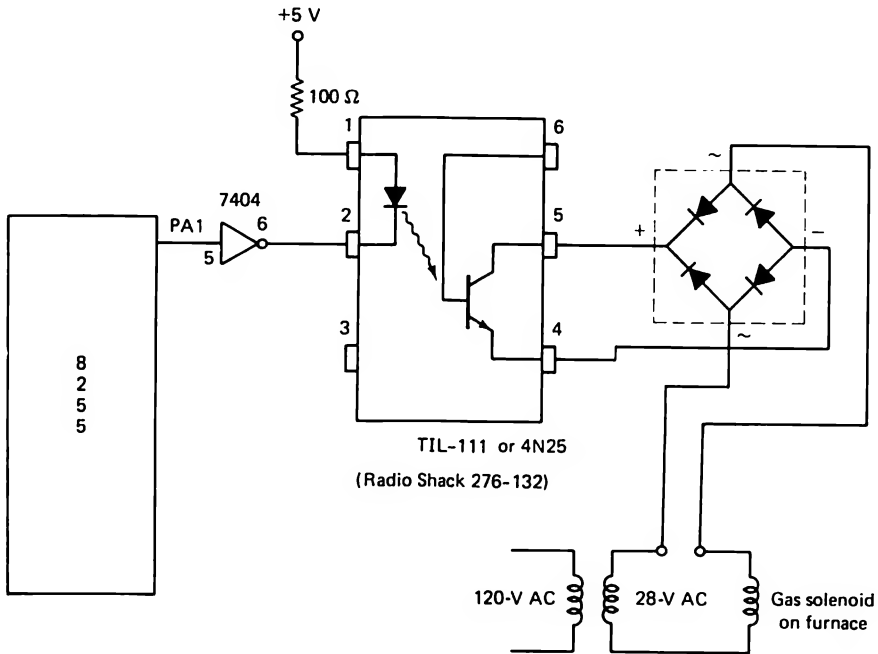


Figure 8-5 Computer-controlled solid-state furnace controller. The four diodes enclosed within the dashed lines are available in a single 4-pin bridge rectifier package (for example, Radio Shack 276-1173).

This means there is no danger of high voltage (120 V AC, for example) from the control circuit inadvertently getting into the controlled circuit and computer.

An Apple interface circuit is shown in Fig. 8-7. The transmitter is controlled by bit PA0 of the 8255. When this bit is a 1, the 40-kHz signal developed by the 7404 oscillator is passed to the output transducer. Because the

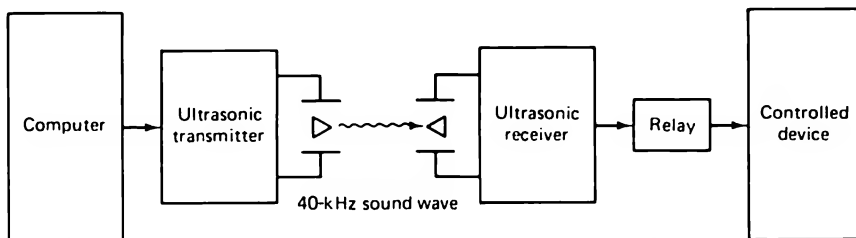


Figure 8-6 Ultrasonic computer interface. The transmitter, when activated by the computer, generates a high-frequency sound wave. When detected by the receiver, this signal can be used to activate a relay and turn on or off the controlled device.

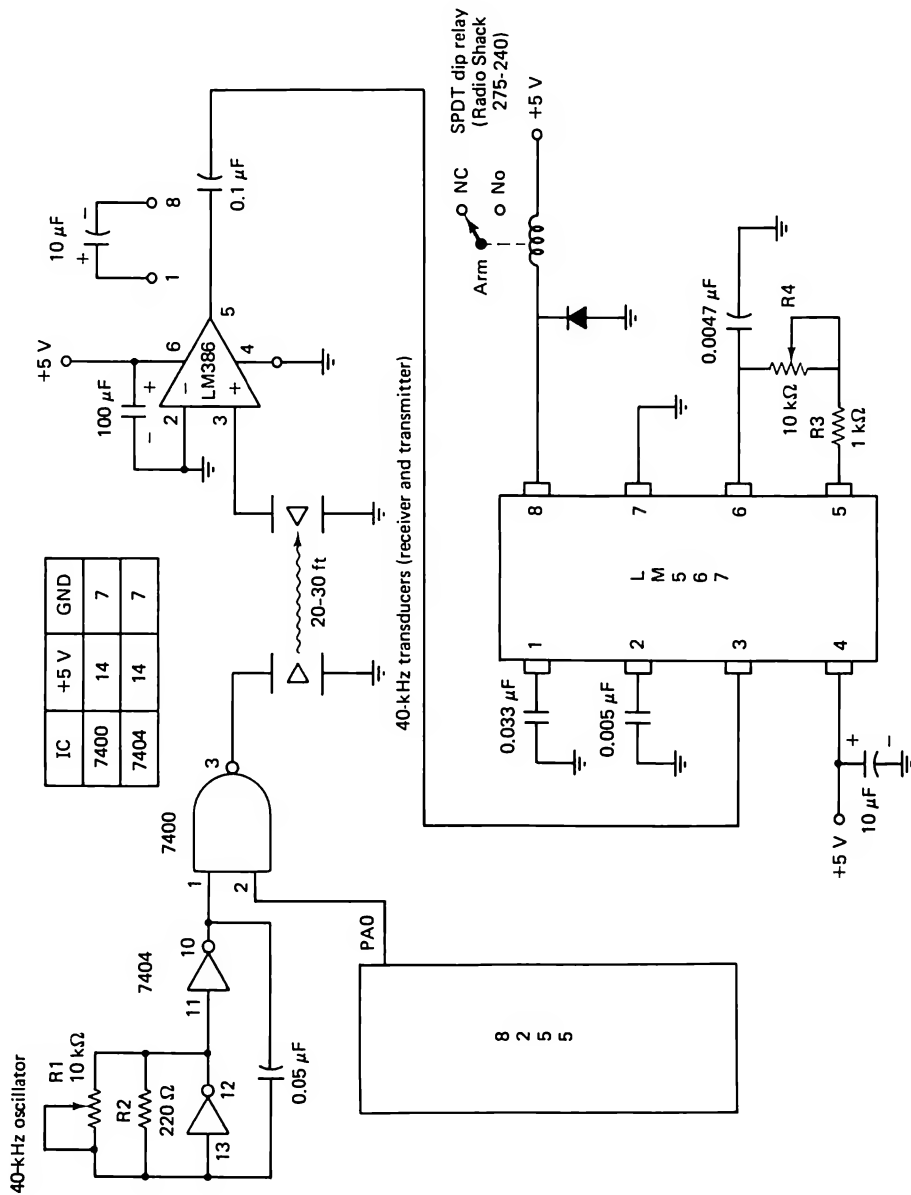


Figure 8-7 Apple ultrasonic interface. The transmitter and receiver communicate over a 40-kHz ultrasonic sound path. The 40-kHz transducers are available from The Micromint Inc., 561 Willow Avenue, Cedarhurst, NY 11516.

transducer is optimized for operation at 40 kHz, R1 should be adjusted to produce 40-kHz oscillations at pin 3 of the 7400.

The receiving transducer detects the 40-kHz signal and passes it to the LM386 eight-pin integrated amplifier. Because the received signal is very tiny (a few millivolts at 20 to 30 ft), this amplifier is a necessity for distances greater than 3 or 4 ft. The gain of the amplifier is fixed at 20 but may be increased to 200 by adding a 10- μ F capacitor between pins 1 and 8. This may be needed for longer distances.

The "heart" of the receiver is the LM567 *tone decoder*. When this IC detects the presence of the 40-kHz tone at its input on pin 3, the output at pin 8 switches to ground. What's more, this output pin has the capability of sinking 100 mA of DC current, making it ideal as a relay driver. The relay contacts can be connected to a 120-V AC lamp or any other AC or DC appliance. The relay specified is rated for 125-V AC at 1 A, but higher current relays could be substituted.

The interface in Fig. 8-7 may appear to provide the ultimate in isolation, but it does have some limitations. The ultrasonic beam produced by the transmitter is easily blocked by a chair, a couch, or your own body (perhaps this suggests a use as a burglar alarm?) and it will not penetrate walls or windows. This means that the transmitter and receiver must generally be in the same room.

Because the circuit described here only transmits an ON or OFF message, it is not too practical to have more than one receiver. This is because all receivers would pick up the same message.

BSR LTD manufactures a complete remote control system that allows placement of up to 16 separate receivers anywhere in your home. This system is similar to the one described here but uses ordinary house wiring to carry specially encoded messages to all receivers. An excellent article describing an interface between this system and the Radio Shack Model I TRS-80 is described by Steve Ciarcia, "Computerize a Home," *BYTE*, January 1980, p. 28.

A similar system is offered by Mountain Hardware Inc., 300 Harvey West Blvd., Santa Cruz, CA 95060. Called the INTROL X-10, this unit plugs into one of the Apple peripheral connectors and allows ON/OFF control of lights and appliances in your home.

Using the Game I/O Connector

The game I/O connector was discussed in Experiment 7. Figure 7-10 details the pin connections of this 16-pin socket. Apple refers to the four output pins (OUT0 through OUT3) as *annunciator* outputs. They provide standard TTL levels.

Each output can be turned ON or OFF by referencing the appropriate memory location. Table 8-1 lists the addresses associated with each output. Note that this technique is quite different from the output ports we have been discussing. For example, to cause pin 15 (OUT0) of the game I/O connector to go high, we would use either of the BASIC commands: POKE 49241,Y or Y=PEEK(49241). The state of the RD/ $\overline{\text{WR}}$ line is unimportant (that's why a PEEK or a POKE command can be used). It is only necessary to force the appropriate address onto the bus. For the same reason, the data used in these commands is unimportant (a POKE 49241,1 or a POKE 49241,0 will both cause OUT0 to go high).

It should be clear that most of the output interfaces we have been discussing can be implemented through the game I/O connector. The relay control circuit in Fig. 8-3, for example, is redrawn in Fig. 8-8 to accommodate the game I/O connector. A BASIC program to monitor the magnetic switch wired to IN0 and activate the relay at OUT0 is

```

10 HOME
20 POKE 49240,Y           :REM MAKE SURE RELAY IS OFF
30 IF PEEK(49249)>127 THEN 20 :REM WAIT FOR OPEN DOOR
40 POKE 49241,Y           :REM THIS TURNS ON RELAY
50 END

```

The main disadvantage to using the annunciator outputs is that only four lines are provided and it will take four separate commands to define each line. Compare this to a standard 8-bit output port in which one command immediately defines all eight lines. Nevertheless, it can be advantageous to use the game I/O connector when it is desired to control single lines with minimum hardware.

TABLE 8-1 ADDRESSES OF THE FOUR ANNUNCIATOR OUTPUTS

Output number	Condition	Address
0	OFF	49240
	ON	49241
1	OFF	49242
	ON	49243
2	OFF	49244
	ON	49245
3	OFF	49246
	ON	49247

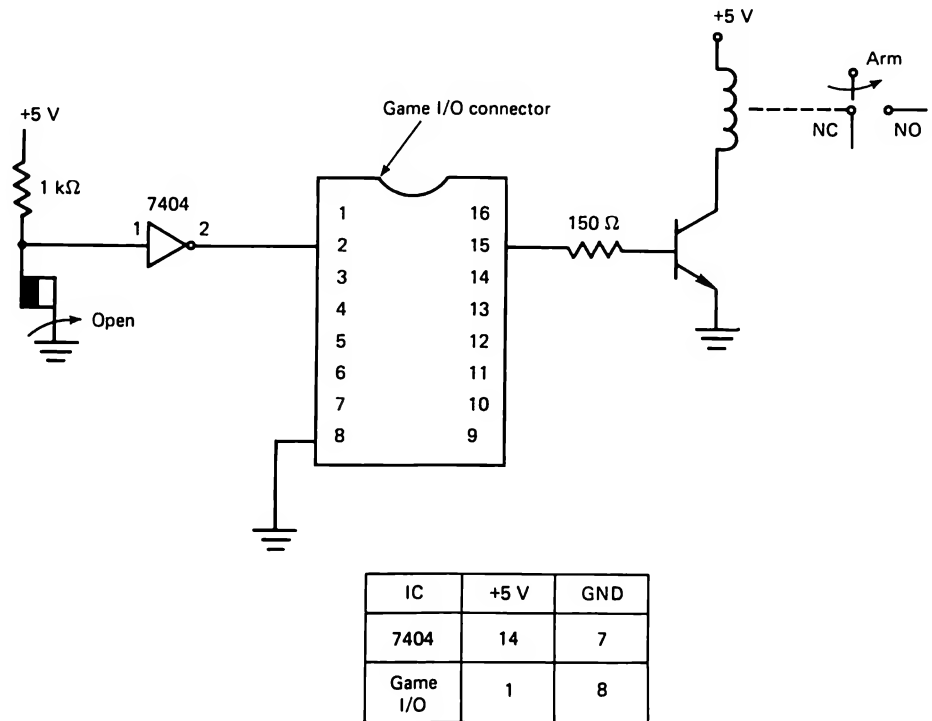


Figure 8-8 Using the game I/O connector to control a mechanical relay. Four output lines are available at this connector.

PROCEDURE

Step 1. Refer to Fig. 8-3 and wire this circuit on your breadboard. If you still have the hardware from Experiment 7 in place, remove the light and temperature sensors but leave the magnetic switch in place. The 8255 wiring is unchanged. Be sure to wire the relay properly by locating the coil, arm, and normally closed (NC) and normally open (NO) contacts. Connect the moving relay arm to +12 V (pin 21 of the socket connector).

Question 8-1. Write a simple program to test this circuit.

Step 2. Connect R2 and the LED to the NC contact and again run the test program from Question 8-1. Note the difference in operation.

Question 8-2. With the circuit wired as in step 2, must the relay be ON or OFF to turn on the LED?

Step 3. Refer to Fig. 8-4 and wire this solid-state relay interface and piezoelectric buzzer to the 8255. The 7404 should already be on your breadboard.

Question 8-3. The test program used for Question 8-1 will also work for this circuit. To what should the **POKE** command be changed?

Step 4. If you still have the magnetic switch interfaced to your computer, write a program to accomplish the following:

1. If the magnets are engaged, turn **ON** the relay and LED.
2. If the magnets are disengaged, turn **OFF** the relay and LED but cause the buzzer to switch on and off rapidly (an alarm condition).

A solution is provided at the end of this experiment.

Note. The following steps involve the ultrasonic interface shown in Fig. 8-7. As explained in the "Discussion," optimum performance occurs when the ultrasonic frequency is set to 40 kHz. This requires an *oscilloscope*, *frequency counter*, or an accurate *signal generator* that can be substituted for the 7404 oscillator.

Step 5. Remove the solid-state and mechanical relay circuits built in the first four steps. Now add a 7400 to your breadboard and wire the ultrasonic transmitter shown in Fig. 8-7. Use two of the extra inverters in the 7404 already on your breadboard.

Step 6. For testing purposes enable this circuit with the following in the immediate mode:

POKE 49395,137: POKE 49392,1

Now, using whatever means are available to you, adjust R1 until the signal at pin 3 of the 7400 is 40 kHz.

Note. The ultrasonic receiver in Fig. 8-7 should ideally be built on a second breadboard located remote to the transmitter. If this is not possible, wire it on your breadboard together with the transmitter. If you plan to test the circuit beyond distances of 2 to 3 ft, you will have to use the LM386 amplifier. If not, the amplifier can be skipped and the receiving transducer's output connected directly to pin 3 of the LM567.

Step 7. With the test program from step 6 holding the transmitter on, adjust the receiver to 40 kHz by adjusting R4 until the relay clicks on reliably.

Step 8. You may wish to connect an AC light to the relay, some other AC device, or the piezoelectric buzzer. Now run the program given in

the solution to Question 8-1. Some experimentation with the position of the two transducers is required for best performance.

Step 9. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

Note. The 8255 portion of this circuit will be used again in Experiment 9.

SOLUTIONS TO QUESTIONS

8-1.

```

10 HOME
20 POKE 49395,137      :REM INIT PPI, PORTS A,B ARE OUTPUTS,
                        C AN INPUT
30 INPUT "DO YOU WANT THE RELAY ON OR OFF ";A$
40 IF A$="ON" THEN POKE 49392,1: GOTO 30
50 POKE 49392,0: GOTO 30

```

8-2. Off

8-3. POKE 49392,2

Step 4 (solution)

```

10 HOME
20 POKE 49395,137      :REM INIT PPI
30 GOSUB 500: IF B(0)=1 THEN POKE 49392,1: GOTO 30
40 REM SOUND THE ALARM
50 POKE 49392,2        :REM BUZZER ON
60 FOR J=1 TO 50: NEXT J      :REM WAIT
70 POKE 49392,0        :REM BUZZER OFF
80 FOR J=1 TO 50: NEXT J      :REM WAIT
90 GOTO 30
500 Y=PEEK(49394)
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN

```

EXPERIMENT 9

INTERFACING A DIGITAL-TO-ANALOG CONVERTER

Note. You must have a 0-5 V voltmeter to monitor the analog output voltage of the converter circuit in this experiment.

OVERVIEW

In this experiment you will interface an MC1408 *digital-to-analog converter* to the Apple. The circuit will be calibrated to provide a 0- to 4.98-V programmable power supply in 20-mV steps. BASIC control software will be developed.

OBJECTIVES

The key points to be learned from this experiment are:

1. A digital-to-analog converter receives a digital word and converts it to a scaled analog voltage, 0 to 5 V, for example.
2. An n -bit converter will produce voltage steps of size equal to the full-scale output voltage divided by 2^n . A typical step size is 20 mV for an 8-bit converter.
3. A digital-to-analog converter is interfaced to a microcomputer through a normal output port.
4. Because BASIC is very slow, machine language is required when using the digital-to-analog converter for complex waveform generation (music or speech, for example). BASIC is adequate for applications such as real-time control of mechanical devices or a programmable power supply.

PARTS LIST

- 1 7404 hex inverter
- 1 8255 programmable peripheral interface (Jameco DP8255)
- 1 MC1408 (see the text) (Jameco MC1408L8)
- 1 LM1458 dual operational amplifier (Radio Shack 276-038)
- 1 LED
- 1 270- Ω resistor (red-violet-brown)
- 1 180- Ω resistor (brown-gray-brown)
- 2 1-k Ω resistors (brown-black-red)
- 1 10-k Ω potentiometer (pot)
- 1 100-pF capacitor
- 1 0-5 V voltmeter

DISCUSSION: DIGITAL AND ANALOG OUTPUTS

In Experiments 7 and 8 we discussed digital interfacing techniques. The circuits in these experiments produced discrete ON/OFF output levels. Input devices were regarded to switch ON or OFF when they crossed some preset reference level. In this experiment and the next we do not restrict our inputs or outputs to these discrete levels. Instead, the whole continuum of values between 0 V and some full-scale value will be allowed. If we wish to output 3.73 V or 1.02 V, our circuit will be capable of doing it.

How is this accomplished? In this experiment we introduce a new integrated circuit called the *digital-to-analog converter* or DAC. This IC is capable of accepting an n -bit digital input word and producing an analog output voltage scaled to the magnitude of this binary word. In Experiment 10 we will interface an *analog-to-digital converter* (ADC). With this circuit, an input voltage (proportional to temperature, light, pressure, etc.) can be converted to a digital input word and then processed by the computer.

Digital-to-Analog Techniques

Figure 9-1 illustrates a hypothetical DAC. The circuit receives a binary input that is used to turn on or off a number of weighted voltage sources. A summing amplifier then adds all inputs to produce the net output voltage. For example, the circuit in Fig. 9-1 will produce a 5-V output when the input word is 0101 (the 4-V and 1-V sources are switched on).

The simple DAC in Fig. 9-1 has 4 bits and therefore 16 steps from 0 V (0000) to 15 V (1111). The size of one step is always equal to the contribu-

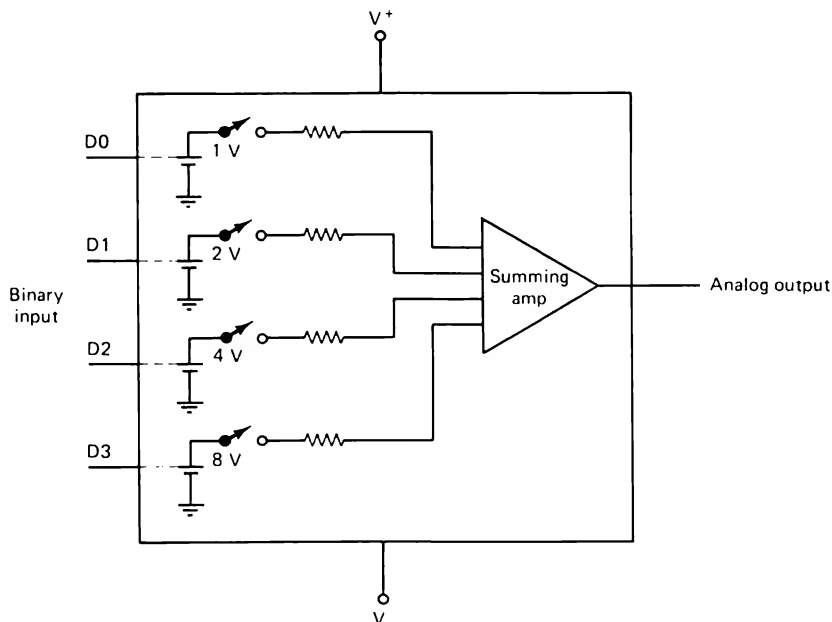


Figure 9-1 Hypothetical 4-bit DAC. The four binary inputs control the four switches. Note that each voltage source is weighted according to its bit position. Maximum output occurs when the binary input is 1111 and produces a 15-V (1 V + 2 V + 4 V + 8 V) output.

tion of the least significant bit, 1 V in this case. Knowing the step size, we can predict the output voltage for any binary input as:

$$(\text{decimal input value}) \times (\text{step size}) = \text{analog output voltage} \quad (9-1)$$

The *full-scale output voltage* of the circuit in Fig. 9-1 is 16 V even though the output can never reach this value! This is because the most significant bit contributes one-half of full scale, the next bit one-fourth, the next one-eighth, and so on. An infinite number of bits is required to actually achieve the full-scale output voltage. For this reason, maximum output is always one step below full scale.

For an n -bit DAC the full-scale output voltage is

$$V_{FS} = 2^n \times \text{step size} \quad (9-2)$$

Example 9-1

For the DAC in Fig. 9-1 determine the full-scale output voltage and output voltage for a 1111 binary input. Assume a 1-V step size.

Solution Applying Eq. (9-2), we get

$$V_{FS} = 2^4 \times 1 \text{ V} = 16 \text{ V}$$

and when the binary input is 1111 (15_{10}), we have

$$V_{\text{out}} = 15 \times 1 \text{ V} = 15 \text{ V}$$

Often, the full-scale output voltage can be adjusted with external components to whatever value you desire. Then the maximum output voltage will be one step less than this value. This type of DAC is referred to as a *multiplying* digital-to-analog converter.

The MC1408

Figure 9-2 is a data sheet and block diagram of the Motorola MC1408 8-bit DAC. This IC is available in several varieties. The MC1408 series will operate from 0 to 75°C, and the MC1508 will operate from -55 to 125°C (the military temperature range). In addition, the MC1408 is subdivided into three accuracy levels. The MC1408L-8 has 8-bit accuracy, the MC1408L-7 has 7-bit accuracy, and the MC1408L-6 has only 6-bit accuracy.

Unlike the hypothetical DAC in Fig. 9-1, the MC1408 uses a *weighted current* switching scheme to produce an output current that is proportional to the 8-bit binary input. Referring to Fig. 9-2, a 2-mA reference current is established at pin 14, which becomes the full-scale output current of the device.

We can rearrange Eq. (9-2) to determine the value of one current step. Substituting I_{FS} (full-scale current) for V_{FS} , we obtain

$$\text{step size} = \frac{I_{\text{FS}}}{2^n} = \frac{2 \text{ mA}}{256} = 7.8125 \mu\text{A}$$

This means that maximum output current will be

$$255 (11111111) \times 7.8125 \mu\text{A} = 1.992 \text{ mA}$$

The absolute accuracy of these values depends on the part used and the stability of the reference current. An MC1408L-6 will be accurate to 1 part in 128 ($\pm 0.78\%$). The MC1408L-8 will be accurate to 1 part in 512 ($\pm 0.19\%$). These numbers represent an accuracy to $\pm \frac{1}{2}$ a least significant bit.

The *transfer characteristics* in Fig. 9-2 illustrate the output current versus binary input. Note that a binary input of 128 (10000000) will provide an output current of $128 \times 7.8125 \mu\text{A} = 1 \text{ mA}$ or half of full scale (128 is half of the 256 possible steps).

Interfacing the MC1408 to the Apple

When interfacing the MC1408 DAC to the Apple, three requirements must be met.

1. An 8-bit output port must be provided. The MC1408 has no internal latches to save the input binary word.

ORDERING INFORMATION

Device	Temperature Range	Package
MC1408L6	0°C to +75°C	Ceramic DIP
MC1408L7	0°C to +75°C	Ceramic DIP
MC1408L8	0°C to +75°C	Ceramic DIP
MC1408P6	0°C to +75°C	Plastic DIP
MC1408P7	0°C to +75°C	Plastic DIP
MC1408P8	0°C to +75°C	Plastic DIP
MC1508L8	-55°C to +125°C	Ceramic DIP

MC1408
MC1508

Specifications and Applications Information

EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

designed for use where the output current is a linear product of an eight-bit digital word and an analog input voltage.

- Eight-Bit Accuracy Available in Both Temperature Ranges
Relative Accuracy: $\pm 0.19\%$ Error maximum
(MC1408L8, MC1408P8, MC1508L8)
- Seven and Six-Bit Accuracy Available with MC1408 Designated by 7 or 6 Suffix after Package Suffix
- Fast Settling Time - 300 ns typical
- Noninverting Digital Inputs are M TTL and CMOS Compatible
- Output Voltage Swing -- +0.4 V to -5.0 V
- High Speed Multiplying Input
Slew Rate 4.0 mA/ μ s
- Standard Supply Voltages: +5.0 V and -5.0 V to -15 V

EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

SILICON MONOLITHIC
INTEGRATED CIRCUIT



L SUFFIX
CERAMIC PACKAGE
CASE 620



P SUFFIX
PLASTIC PACKAGE
CASE 648

FIGURE 1 - D to A TRANSFER CHARACTERISTICS

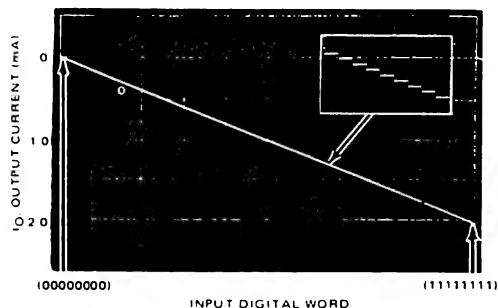
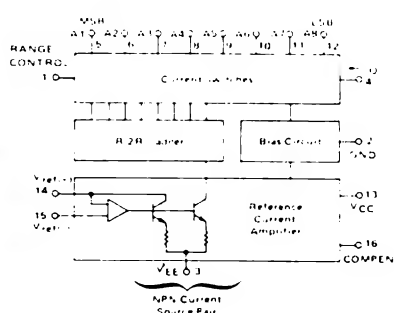


FIGURE 2 - BLOCK DIAGRAM



TYPICAL APPLICATIONS

- Tracking A-to-D Converters
- Successive Approximation A-to-D Converters
- 2 1/2 Digit Panel Meters and DVM's
- Waveform Synthesis
- Sample and Hold
- Peak Detector
- Programmable Gain and Attenuation
- CRT Character Generation
- Audio Digitizing and Decoding
- Programmable Power Supplies
- Analog-Digital Multiplication
- Digital-Digital Multiplication
- Analog-Digital Division
- Digital Addition and Subtraction
- Speech Compression and Expansion
- Stepping Motor Drive

Figure 9-2 Block diagram and transfer curves for the MC1408 DAC.
(Courtesy of Motorola Semiconductor Products Division.)

2. A 2-mA *reference current* must be applied to pin 14 of the DAC. This current determines absolute accuracy and stability of the interface.
3. A *current-to-voltage converter* circuit must be used on the DAC output if an output voltage is desired instead of current.

A circuit that accomplishes these requirements is shown in Fig. 9-3. Note the following about this circuit:

1. The 8255 is programmed to make port A an output. Data sent to this port will be converted to a weighted current by the DAC.
2. The 2-mA reference current is established by the R1R2 voltage divider, which develops 2 V ($180\ \Omega / (180\ \Omega + 270\ \Omega) \times 5\text{ V}$) on pin 6 of the LM1458. This op-amp is used as a *voltage follower* passing the 2 V to R3, which establishes the 2 mA ($2\text{ V} / 1\text{ k}\Omega$).
3. The other half of the LM1458 provides the *current-to-voltage* conversion. Output current, I_o , flows through R4, which can be adjusted to yield any desired full-scale output voltage within the saturation limits of the op-amp.

Example 9-2

What value should R4 be adjusted to if full-scale output is to be 5.0 V?

Solution Because the full-scale output current is 2.0 mA, $R4 = 5\text{ V} / 2\text{ mA} = 2.5\text{ k}\Omega$.

Example 9-3

What is the maximum output voltage for the circuit in Fig. 9-3? Assume that $V_{FS} = 5.0\text{ V}$.

Solution Apply Eq. (9-2) to determine the step size:

$$\text{step size} = \frac{V_{FS}}{2^n} = \frac{5.0\text{ V}}{256} = 19.53\text{ mV}$$

Then when the binary output is 11111111 (255_{10}),

$$V_{\text{out}}(\text{max}) = 255 \times 19.53\text{ mV} = 4.98\text{ V}$$

The results of Examples 9-2 and 9-3 indicate that, under computer control, we should be able to adjust the output voltage of the circuit in Fig. 9-3 to any voltage between 0 and 4.98 V in approximately 20-mV steps.

DAC Software

Controlling the DAC in Fig. 9-3 from BASIC is extremely simple. The only requirement is that the desired binary value be output to port A of the 8255.

Figure 9.3 Eight-bit DAC interface to the Apple. Port A of the 8255 is programmed as an output port and the MC1408 converts the binary output word to an analog voltage. The LM1458 op-amp serves as a voltage-to-current converter and 2-mA voltage-follower reference circuit.

For example, to output 2.5 V, the binary code 10000000 (128_{10}) should be output.

```

10 POKE 49395,137           :REM INIT PPI, A,B=OUTPUTS, C=INPUT
20 POKE 49392,128           :REM THIS IS PORT A
30 END

```

The DAC circuit in Fig. 9-3 must be *calibrated* before it can produce accurate output voltages. This can be accomplished by running the test program just given and adjusting R4 until $V_{out} = 2.50$ V.

It is also very important that the *reference current* be exactly 2 mA for absolute accuracy. This is established by the voltage-follower circuit connected to pin 14 of the DAC. If the value of either resistor R1 or R2 is incorrect, or the 5-V source is not exactly 5 V, errors will occur. Although the reference in Fig. 9-3 is suitable for our testing purposes, more stable precision references should probably be considered for serious applications.

Example 9-4

Develop a general formula that can be applied to the circuit in Fig. 9-3 to determine the proper binary code for any desired output voltage between 0 and 4.98 V. What code should be used to output 3.65 V?

Solution We need only determine what fraction of the full-scale output is desired and then scale this result to a number between 0 and 255.

$$\frac{V_{out}}{5.0 \text{ V}} = \frac{X}{256}$$

where X is the unknown code. Solving for X yields

$$X = \frac{V_{out}}{5.0 \text{ V}} \times 256$$

Now, to produce 3.65 V,

$$X = \frac{3.65 \text{ V}}{5.0 \text{ V}} \times 256 = 186.88$$

A POKE 49392,187 should be used.

One limitation to the circuit in Fig. 9-3 is that the output current is limited to 20 to 25 mA by the op-amp. This current capability can be increased fairly simply by the addition of a “*pass transistor*,” as shown in Fig. 9-4. Operation is identical to Fig. 9-3, but the load current will now pass through Q1 instead of the op-amp. Suitable heat sinking for Q1 will allow currents in excess of 1A.

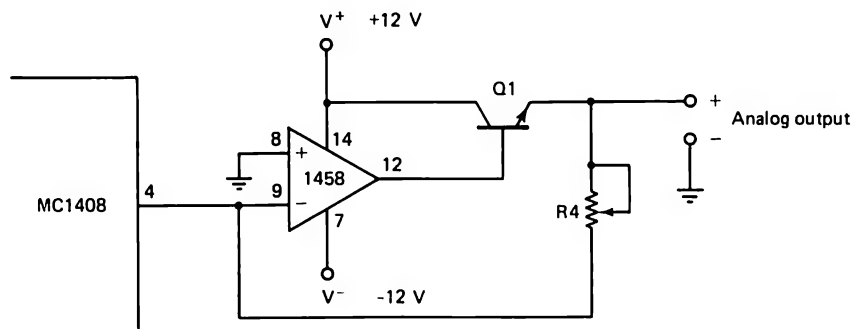


Figure 9-4 This addition to the circuit in Fig. 9-3 increases the current capabilities to several hundred milliamperes or more, depending on the transistor used and heat sinking (if any).

Other DACs

When designing a digital-to-analog interface circuit, several considerations should be made. Among these are:

1. **Bits of resolution.** DACs are commonly available with 8 to 12 bits. The more bits the converter has, the smaller the step size and the greater the resolution.
2. **Internal reference.** All DACs require a stable reference current or voltage. On most this must be supplied externally adding cost and complexity to the design.
3. **Internal buffering.** Unless the DAC has an internal buffer (latch), one must be designed into the system. Converters with greater than 8 bits require double buffering.
4. **Settling time.** This is the time required for the output voltage to settle and remain around the final value. This in turn indicates how rapidly the digital input may vary.
5. **Linearity.** This defines the deviation from the straight-line transfer characteristic shown in Fig. 9-2. This is often expressed as a percentage or fraction of a least significant bit ($\pm \frac{1}{2}$ LSB equals $\pm \frac{1}{512}$ or 0.19% for an 8-bit converter, for example).
6. **Internal amplifier.** The output of most DACs is a current that must be converted to a voltage. If this conversion is not provided internally, it also must be designed into the system.

Table 9-1 presents a summary of this information for several commonly available DACs.

TABLE 9-1 EXAMPLES OF COMMERCIALY AVAILABLE 8-, 10-, AND 12-BIT DACs

Part number	Resolution (bits)	Manufacturer	Supplies	Linearity (%)	Settling time (ns)	Internal ref.	Internal buffer	Internal amp.	DIP pins
DAC0800	8	National Semiconductor	+4.5 to +18 V	± 0.19	100	No	No	No	16
MC1408L8	8	Motorola	+5 V, -15 V	± 0.19	300	No	No	No	16
MC1408L7	8	Motorola	+5 V, -15 V	± 0.39	300	No	No	No	16
MC3410	10	Motorola	+5 V, -15 V	± 0.05	250	No	No	No	16
DAC1000	10	National Semiconductor	+5 to 15 V	± 0.05	500	No	Yes	No	24
DAC-02	10	Precision Monolithics	± 15 V	± 0.1	1500	Yes	No	Yes	18
DAC1200	12	National Semiconductor	± 15 V, 5 V	± 0.012	1500	Yes	No	Yes	24
AD563	12	Analog Devices	+5 V to 15 V, -15 V	± 0.012	1200	Yes	No	No	24

PROCEDURE

Step 1. Refer to Fig. 9-3 and if you have not already done so, wire the 8255 and 7404 inverter on your breadboard.

Step 2. You can make sure that this portion of the interface circuit works by wiring the simple *logic-level indicator* circuit shown in Fig. 9-5. Touch its input to the port A outputs of the 8255 and run the following program with N=0 and then 255.

```

10 HOME: POKE 49395,137           :REM INIT PPI
20 N=0
30 POKE 49392,N                   :REM OUTPUT N TO PORT A
40 END

```

Question 9-1. What should you see with the logic-level indicator for these two values of N?

Step 3. When your hardware passes the test in step 2, continue wiring the circuit in Fig. 9-3 by adding the MC1408 and LM1458 op-amp. Connect -12 V (pin 30 of the socket connector) to pin 3 of the MC1408 and pin 7 of the LM1458. Connect +5 V to pin 13 of the MC1408 and +12 V (pin 21 of the socket connector) to pin 14 of the LM1458.

Step 4. Calibrate the DAC by following the test procedure given under "DAC Software" in the "Discussion" section.

Question 9-2. Test the interface by running the calibration program from step 4 with line 20 changed so that you output 24_{10} , 97_{10} , 163_{10} , and 237_{10} . What output voltage do you measure for each of these values?

Step 5. Write a program to make the output voltage ramp from 0 to +5 V, dropping abruptly back to 0 V, and repeating this cycle. This program should generate a *sawtooth* waveform.

Question 9-3. Modify the program in step 5 to produce a *triangular* waveform. The output should ramp from 0 to +5 V and back down to 0 V.

Question 9-4. What is the *period* of the triangular waveform generated in Question 9-2? Is it realistic to use a BASIC control program for music generation?

Step 6. Develop a BASIC program that will make the DAC perform as a *programmable power supply*. The Apple should prompt you for an

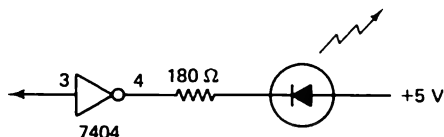


Figure 9-5 Simple logic-level tester using one of the spare inverters in the 7404. A logic 1 on pin 3 will turn ON the LED, a 0 will turn it OFF.

output voltage between 0 and 4.98 V and then determine the proper decimal code to output to the DAC to produce that voltage. (*Hint*: Study Example 9-4 for a general formula that can be used to calculate the proper code. Also note that the computer will *truncate* noninteger results, so you may want to compensate for this in your program.) A solution is provided at the end of this experiment.

Step 7. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

Note. The 8255 will not be used in Experiment 10.

SOLUTIONS TO QUESTIONS

9-1. $N=0$; all outputs are low (LED OFF at each pin). $N=255$; all outputs are high (LED ON at each pin).

9-2. 0.47 V, 1.89 V, 3.18 V, 4.63 V

9-3. One possible solution:

```

10 POKE 49395,137           :REM INIT PPI
20 FOR J=0 TO 255           :REM RAMP UP
30 POKE 49392,J
40 NEXT J
50 FOR J=254 TO 1 STEP -1   :REM RAMP DOWN
60 POKE 49392,J
70 NEXT J
80 GOTO 20

```

9-4. On the Apple II the period is approximately 3 s (0.3 Hz). Because BASIC takes so long to interpret one program line, it is not practical to generate music with this technique.

Step 6 (solution)

```

10 HOME
20 POKE 49395,137           :REM INIT PPI
30 VTAB(1)
40 INPUT "ENTER OUTPUT VOLTAGE (0-4.98V): ";V
50 IF V>4.98 OR V<0 THEN 30
60 V1=(V/5)*255             :REM SCALE V (0-255)
70 V2=INT(V1)               :REM FIND INTEGER VALUE
80 IF V1-V2>=.5 THEN V2=V2+1 :REM ROUND OFF
90 POKE 49392,V2            :REM OUTPUT VALUE
100 VTAB(10): PRINT "VOUT= ";V;" VOLTS "
110 VTAB(12): PRINT "DECIMAL CODE= ";V2;" "
120 GOTO 30

```

EXPERIMENT 10

INTERFACING AN ANALOG-TO-DIGITAL CONVERTER

OVERVIEW

In this experiment you will interface the National Semiconductor ADC0809 *analog-to-digital converter* to the Apple. An eight-channel *digital voltmeter* will be programmed. A BASIC program to convert the Apple into a *storage oscilloscope* using high-resolution graphics is given.

OBJECTIVES

The key points to be learned from this experiment are:

1. An analog-to-digital converter is used to convert an analog input voltage to an n -bit digital word.
2. One common conversion technique is to use a digital-to-analog converter in a feedback loop with a digital counter or computer.
3. Two methods for controlling this analog-to-digital converter result in the *successive approximations* and *tracking converter* designs.
4. Fully integrated data acquisition ICs are readily available and may easily be interfaced to the Apple computer.

PARTS LIST

- 1 7402 quad two-input NOR gate
- 1 7404 hex inverter

- 1 LM1458 dual operational amplifier (Radio Shack 276-038)
- 1 ADC0809 8-channel analog-to-digital converter (Jameco ADC0809CCN)
- 1 LM334 temperature sensor (Radio Shack 276-1734)
- 1 6.2-V zener diode (optional, see text)
- 1 220- Ω resistor (red-red-brown)
- 3 1-k Ω resistors (brown-black-red)
- 1 10-k Ω resistor (brown-black-orange)
- 1 330-k Ω resistor (orange-orange-yellow)
- 1 10-k Ω potentiometer (pot)
- 1 50-k Ω potentiometer (pot)
- 1 680-pF capacitor
- 2 10- μ F capacitors

DISCUSSION: CONVERTING AN ANALOG VOLTAGE TO A DIGITAL INPUT

This experiment will explore the other half of analog interfacing begun in Experiment 9: namely, interfacing an analog input voltage to the Apple. Experiment 7 has already shown how non-TTL level signals may be interfaced, but these voltages were of an ON/OFF nature. In this experiment the input voltage will be allowed to have any value between 0 V and some full-scale value.

The circuit required to do this conversion is called an *analog-to-digital converter*, or ADC. A block diagram of a typical converter interface is shown in Fig. 10-1. The ADC receives an input voltage which must be between 0 V and its full-scale value (established by V_{REF}) and converts this voltage to an n -bit digital word (8 bits in this example). A select pulse is then used to enable the tri-state gates of an input port and gate the data into the computer.

Much of the terminology associated with the *digital-to-analog converter* (DAC) also relates to the ADC. For example, an n -bit ADC will only recognize discrete steps of the input voltage. If $n = 4$, there will be 16 unique digital words that can be input from the ADC, representing 16 possible values of the input voltage.

Example 10-1

Refer to the ADC interface in Fig. 10-1 and determine the step size and digital output for a 1.25-V input. Assume that full-scale voltage is 5.0 V.

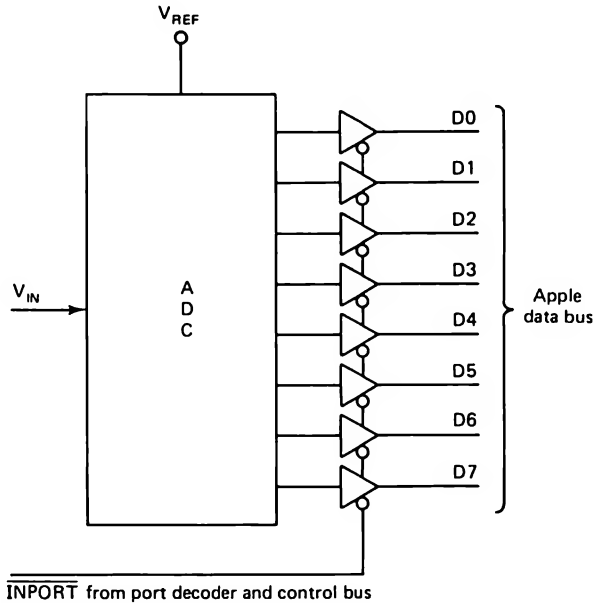


Figure 10-1 Simplified block diagram of an ADC interface. In reality, most ADCs must be “told to make a conversion” before the digital equivalent of the input voltage is available to the computer.

Solution The concept of step size is identical to that encountered with the DAC in Experiment 9. Using Eq. (9-2),

$$\text{step size} = \frac{V_{FS}}{2^n} = \frac{5.0 \text{ V}}{256} = 19.53 \text{ mV}$$

If the input voltage is 1.25 V, the binary output will be

$$\frac{1.25 \text{ V}}{5.0 \text{ V}} \times 256 = 64_{10} = 01000000$$

As this example points out, the ADC is basically treated as a DAC in reverse. You can also see that the full-scale voltage is very critical in determining the range of input voltages and resulting binary output word. For some ADCs this value is preset; for others it may be adjustable with a potentiometer.

One last difference between the DAC and ADC should be pointed out. The DAC produces a steady output voltage as long as its binary input is unchanging. The ADC must generally be told to convert its present analog input to digital. This is referred to as a *conversion*. On some ADC circuits this conversion may take several milliseconds; on others it may take only

tens of microseconds. The point is that the ADC does not do this conversion on its own; it must be told to do so by external hardware or software.

The Flash Converter

Perhaps the simplest way to visualize an analog-to-digital converter is the so-called “flash” converter shown in Fig. 10-2. In this circuit three analog comparators compare the input voltage against each possible step value. The output of a comparator is high when V_{in} is greater than the step value and low when less than this value. A logic gate array is needed to convert the comparator outputs to the standard 1-2-4-8 binary code.

The circuit in Fig. 10-2 is extremely simple because it only provides a 2-bit digital word, but it does illustrate the concept. A major advantage to this type of ADC is the *conversion speed*, which is limited only by the propagation delays of the comparators and logic gates. This may result in conversion times of less than $1\ \mu\text{s}$.

Of course, a 2-bit ADC is not too practical! With 4 V full scale, each step must be 1 V ($4\text{ V}/2^2$). If the circuit is expanded to 8 bits, 256 different voltage levels must be detected and 255 comparators will be needed (none is needed for the 0-V step). This is the major disadvantage of this technique. An n -bit converter will require $2^n - 1$ comparators. And although the technology does exist to put 255 (or more) comparators on a single piece of

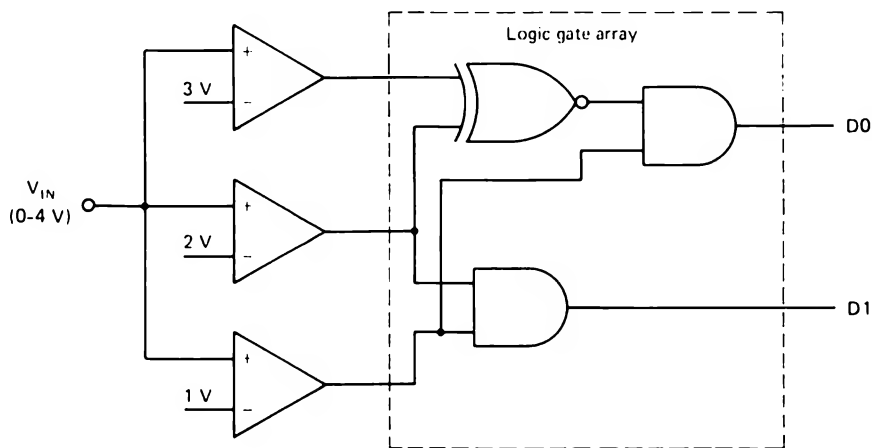


Figure 10-2 Two-bit “flash” converter. The three comparators detect the three possible steps: 1 V, 2 V, and 3 V. The logic gates convert the comparator outputs to standard binary code.

silicon, the cost may be prohibitive. Nevertheless, when high speed is needed, flash converters may offer the only alternative.

The Tracking ADC

The tracking ADC uses a more roundabout technique than the straightforward flash converter. With this technique a DAC is used to output a known voltage. This voltage is compared against the unknown input voltage and the binary code sent to the DAC adjusted up or down as necessary. The DAC output eventually converges on the input voltage with its binary input corresponding to the input voltage. Figure 10-3 illustrates the technique.

The computer program controlling the DAC makes an initial guess at the value of V_{unknown} by outputting a binary code to the DAC. The DAC converts this to a voltage that is compared by the analog comparator against V_{unknown} . As Fig. 10-3 illustrates, if the comparator output is high, the program's guess was too big; if low, it was too small. By connecting the comparator output to an input port, the control program can tell if the current guess needs to be adjusted up or down.

This type of interface is called a *tracking* analog-to-digital converter and can be built with relatively inexpensive components. As you might imagine, it is rather slow and also ties up the computer while making the conversion process. The computer can be taken out of the loop by using the comparator output to control the up/down-counting direction of a *binary counter*. This is illustrated in Fig. 10-4.

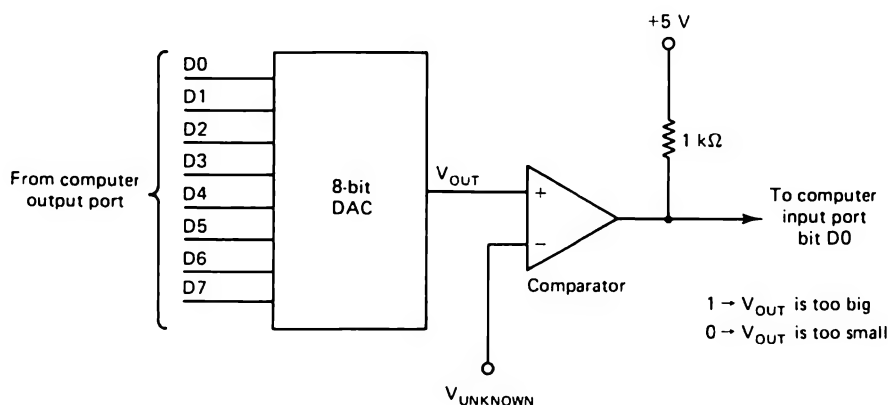


Figure 10-3 Eight-bit ADC suitable for computer control. The unknown input voltage is compared against the DAC's output voltage and the 8-bit output word adjusted up or down until V_{out} and V_{unknown} converge.

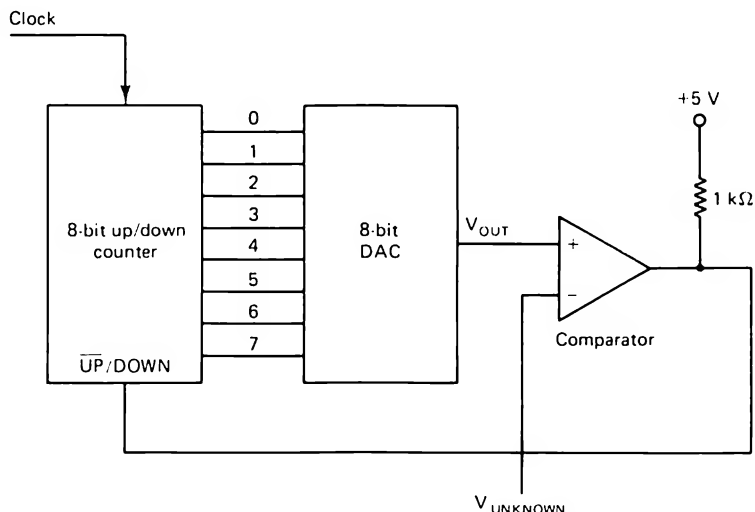


Figure 10-4 Tracking ADC. No computer control is necessary, as the comparator output is used to control the counting direction of an up/down binary counter.

Successive Approximations

The tracking method of analog-to-digital conversion is not really too “smart.” It is analogous to a number-guessing game. You must determine a number your opponent is thinking of by asking if a certain number (your guess) is too high or too low. If the range of values is 0 to 255, the tracking technique has us first guess 128 and then 129 and 130 and 131 and so on until finally arriving at the correct number. A worst-case scenario has us just finding 255 to be the answer, when the input suddenly switches to 0. It will now take 255 more guesses to find this new value! Therefore, in addition to being slow, the tracking ADC has a hard time following analog signals that change value rapidly between extremes.

Continuing with the guessing-game analogy, we might try to come up with a better “guessing algorithm.” Assuming an 8-bit ADC:

1. As a first guess, turn on the most significant bit, 128 in this case.
2. If this guess is too big, turn off this bit; if not, leave it on.
3. Now turn on the next most significant bit, 64 in this case.
4. Repeat this process until all 8 bits have been tested and one conversion is complete.

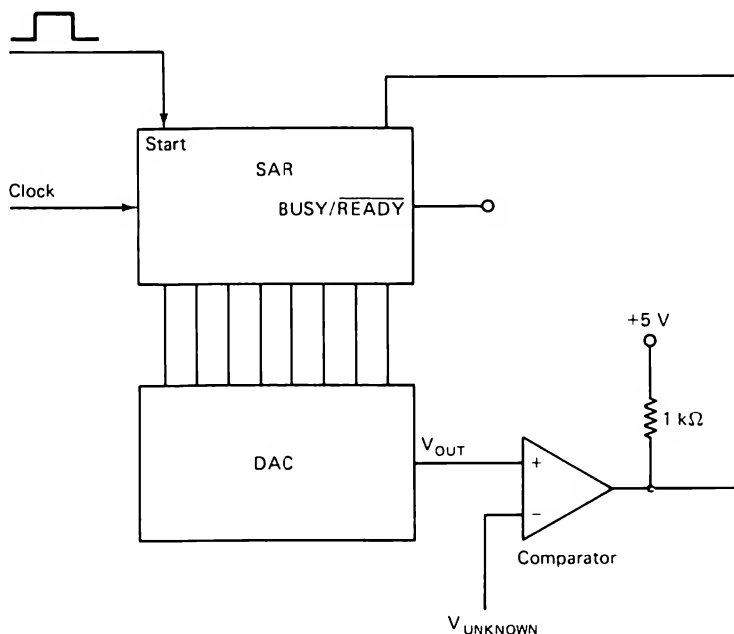


Figure 10-5 Successive approximation ADC circuit. This circuit converges on the proper digital output word much faster than the tracking design.

The advantage to this technique, called *successive approximations*, is that only n tests are needed for an n -bit word compared to $2^n - 1$ (worst case) for the tracking method.

Don't think a computer must be used for this technique either. Figure 10-5 illustrates the use of an SAR (*successive approximation register*) together with a DAC and analog comparator. The SAR can be thought of as a special type of digital counter that follows the algorithm just presented.

Interfacing the ADC0809 to the Apple

It seems to be true in most digital designs today that if you look hard enough, you can find a single IC to accomplish your design goal. The ADC0809 is National Semiconductor's answer to the analog data acquisition problem. Among its features are:

1. Eight separate inputs
2. Internal SAR, DAC, and comparator
3. Tri-state outputs to facilitate microcomputer interfacing
4. Typical conversion time of $100\ \mu\text{s}$

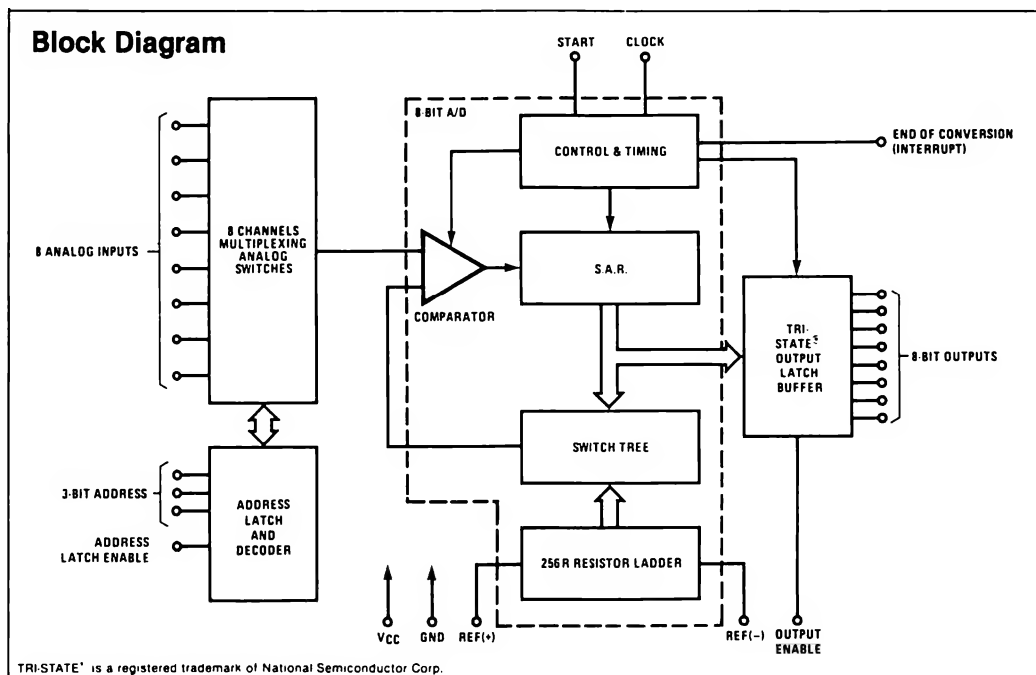


Figure 10-6 Block diagram of the ADC0809 eight-channel analog-to-digital converter. (Courtesy of National Semiconductor.)

Figure 10-6 is a block diagram of this chip. The three address lines allow selection of one of the eight input channels. The conversion process begins by applying an active high pulse (100 to 200 ns) to the start input. The DAC is made up of the *switch tree* and *256R resistor ladder*. This resistor network acts as a voltage divider controlled by the SAR. The comparator compares the switch tree output with the input voltage and the result is applied to the SAR. The typical time for convergence on the proper binary code is 100 μ s. The 8-bit output can be connected directly to a micro-computer data bus and the output enable pin pulsed by an input port enable pulse.

Figure 10-7 illustrates an interface between this chip and the Apple computer. Note the following about this interface:

1. The chip is interfaced directly to the Apple address, data, and control buses; a programmable I/O chip (such as the 8255) is not required.
2. The chip occupies eight addresses corresponding to $\overline{\text{DEV7}}$ low and the eight combinations of A0 through A2. These addresses are 49392 through 49399. Because A3 is not decoded, the circuit will also respond to the address range 49400 through 49407.

3. The conversion is begun by **POKING** the address corresponding to the analog input channel we wish to convert. For example, input 2 maps to address 49394.
4. After 100 μ s, the 8-bit binary equivalent of this channel is available and can be input by **PEEKING** any address that causes $\overline{\text{DEV7}}$ to be low.
5. The clock signal is generated by the 7404 oscillator circuit shown. The clock frequency is approximately 500 kHz.
6. V_{REF} establishes the full-scale input of the ADC. By adjusting this to 5.0 V, one step will equal 19.53 mV (5.0 V/256). The manufacturer recommends that V_{REF} and V_{CC} be matched in this configuration.

Example 10-2

Write a BASIC program to monitor input channel 0 and continually output the voltage value to the Apple screen.

Solution Channel 0 corresponds to address 49392 in Fig. 10-7. One possible program is

```

10 HOME
20 POKE 49392,Y           :REM CONVERT CHANNEL 0
30 VTAB(12):HTAB(8)
40 V=(PEEK(49392)/256)*5  REM SCALE TO 5V
50 PRINT " CHANNEL 0 = " ;V;" VOLTS"
60 GOTO 20

```

PROCEDURE

Step 1. Refer to the Apple ADC interface circuit in Fig. 10-7. Begin wiring this circuit by constructing the 5.0-V V_{REF} and V_{CC} supply circuit. If desired, the 6.2-V general-purpose zener diode may be omitted, or, for accurate work, replaced by a precision temperature-compensated zener diode such as the LM329B (6.9 V). In either case, R1 should be adjusted until the LM1458 output is 5.0 V.

Step 2. Continue wiring the remainder of the circuit, taking normal precautions when handling the ADC0809 MOS device.

Question 10-1. What BASIC command should be given to begin a conversion of the voltage applied to input channel 0?

Note. Because BASIC takes so long to interpret and execute one program line, there is no need to monitor the ADC0809's EOC (end of conversion) output. We can assume that the input voltage has been converted by the time the next program line occurs. If the ADC is controlled with machine language, the EOC output must be monitored.

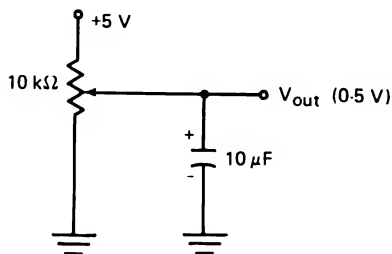


Figure 10-8 Simple 0 to 5 V test voltage for the circuit in Fig. 10-7.

Step 3. The following program converts the Apple into a *digital voltmeter*. The simple circuit in Fig. 10-8 can be used to supply a test voltage to the ADC. With a test voltage applied to pin 26 (IN0), run the following program.

```

10 HOME
20 POKE 49392,Y           :REM CONVERT CHANNEL 0
30 V=(PEEK(49392)/256)*5  :REM SCALE TO 5V
40 VTAB(12): HTAB(18)
50 PRINT V;" VOLTS      "
60 GOTO 20

```

Question 10-2. The program in step 3 displays eight digits of accuracy to the right of the decimal point. This is misleading because the ADC step size is only 0.02 V. Change the program in step 3 so that only two digits to the right of the decimal point are displayed.

Step 4. The ADC0809 has the capability of monitoring eight separate analog inputs. Write a program to monitor the eight input channels continuously and display their values on the screen. A solution is provided at the end of this experiment.

Note. When running this program, be sure that all eight input channels are connected to some voltage between 0 and 5 V.

Note. The ADC circuit and software developed could now be used to monitor eight analog sensors displaying their values on the CRT screen. In the next step we will use the LM 334 *temperature sensor* used in Experiment 7. The interface circuit is shown in Fig. 10-9. By connecting the 220-Ω resistor as shown, only two wires need be run back to the computer. Also note that the distance between the computer and the sensor can be almost any length without affecting the accuracy (the sensor provides a constant current dependent only on temperature). The output voltage across the 10-kΩ resistor will vary 10 mV/°K and at room temperature (70°F), the output voltage should be 2.94 V (294°K). We can convert this voltage to °F as follows:

$$F = \left(((V \times 100) - 273) \times \frac{9}{5} \right) + 32$$

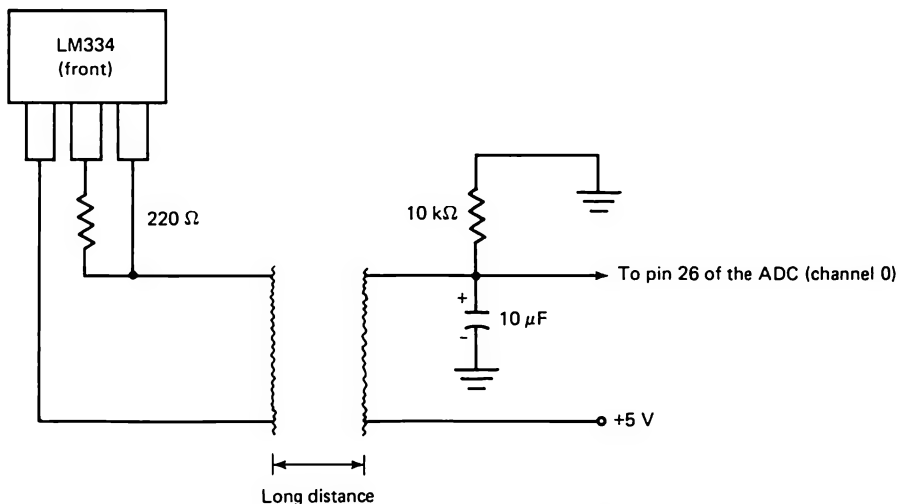


Figure 10-9 The LM334 temperature sensor provides an output voltage that can be converted to degrees Kelvin, Celsius, or Fahrenheit. By connecting the 220-Ω resistor as shown, only two leads are needed for the interface.

Step 5. Connect the LM334 temperature sensor circuit shown in Fig. 10-9 to channel 0 of the ADC. Now change the program given in step 3 as follows:

```

38 T=INT(((V*100)-273)*1.8)+32
40 VTAB(12): HTAB(11)
50 PRINT "T = ";T;" DEGREES"
60 GOTO 20

```

Note. Be careful when interpreting the accuracy of this program. The LM334 output changes 10 mV/°K and the ADC can resolve a 20-mV change. This means that a change of two Kelvin degrees (or 3.6 Fahrenheit degrees) is required before the ADC can detect it. In addition, the accuracy of the LM334 is about $\pm 3\%$, making the temperature display accurate to only 5 to 10°F.

Step 6. The following program uses the *high-resolution graphics* of the Apple to create a graph of voltage versus time. When you run this program, note the following:

1. The screen format is 160 vertical by 280 horizontal.
2. The horizontal scale is in relative time units requested in line 40. Replying 1 to this input will sweep the screen in about 13 s. A reply of 800 will result in approximately one sample each second and sweep the screen in about 5 minutes.

3. The program pauses briefly in line 50 to allow you to set up the input voltage. A beep signifies that the sampling has begun.

Note. A particularly good demonstration can be obtained by monitoring the voltage across a capacitor as shown in Fig. 10-10. Use $T = 100$ for the sampling interval. The characteristic *exponential* charging curve will be displayed.

```

10 HOME : VTAB(12)
20 INPUT "WHICH CHANNEL (0-7) "; C
30 PRINT
40 INPUT "TIME BETWEEN SAMPLES "; T
50 FOR I=1 TO 1000: NEXT I           :REM PAUSE FOR SETUP
60 PRINT CHR$(7)                     :REM BEEP!
70 HGR                               :REM GO TO GRAPHICS
80 HPLOT 0,159 TO 279,159            :REM DRAW AXES
90 X=0                               :REM X=HORIZ POSIT
100 VTAB(21)                         :REM TEXT IS VISIBLE
110 PRINT "CHANNEL = "; C
120 VTAB(23)
130 PRINT "SAMPLE INTERVAL = "; T
140 POKE 49392+C,0                   :REM BEGIN CONVERSION
150 Y=(PEEK(49392)/256) * 160         :REM Y=VERTICAL POSITION
160 Y=160-Y                          :REM INVERT Y VALUE
170 HPLOT X,Y                        :REM PLOT THE POINT
180 FOR I=1 TO T: NEXT I             :REM SAMPLE INTERVAL
190 X=X+1                            :REM NEXT HORIZ
200 IF X=280 THEN END
210 GOTO 140

```

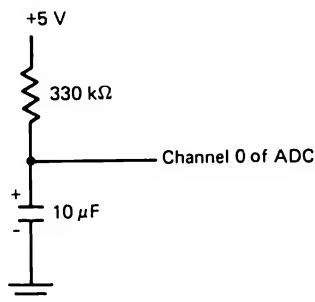


Figure 10-10 Circuit used to test the graphics mode of the ADC interface and program in step 6.

Step 7. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

SOLUTIONS TO QUESTIONS

10-1. POKE 49392,Y

10-2. Add the following lines:

```
35 V1=100*V
36 IF V1-INT(V1)>=.5 THEN V1=V1+1
37 V=(INT(V1))/100
```

Step 4 (solution)

```
10 HOME
20 FOR J=0 TO 7 :REM J=CHANNEL SELECTOR
30 POKE 49392+J,0 :REM CONVERT CHANNEL J
40 V=(PEEK(49392+J)/256) *5 :REM SCALE TO 5V
50 GOSUB 500
60 VTAB(2+3*J): HTAB(9)
70 PRINT "CHANNEL ";J;" = ";V;" VOLTS  "
80 NEXT J
90 GOTO 20
500 V1=100*V
510 IF V1-INT(V1)>=.5 THEN V1=V1+1
520 V=(INT(V1))/100
530 RETURN
```

EXPERIMENT 11

HANDSHAKING I/O

OVERVIEW

In this experiment you will program the 8255 to operate in *mode 1* (strobed input/output). A DIP switch, 74100 latch, and seven-segment display will be interfaced and used to simulate a strobed I/O device. A software seven-segment decoder will be described and tested.

OBJECTIVES

The key points to be learned from this experiment are:

1. With *unconditional* I/O, the microcomputer outputs or inputs data from its peripherals without regard for their BUSY/READY status.
2. *Conditional* I/O requires the computer to monitor the BUSY/READY status flag of the I/O device. This is done using either a *polled* or an *interrupt-driven* technique.
3. Polling is simplest to accomplish, but is slow and inefficient compared to interrupt-driven techniques.
4. When operating in mode 1, the 8255 provides several status indicators and strobe inputs that facilitate a handshaking I/O interface.

PARTS LIST

- | | |
|---|--|
| 1 | 8255 programmable peripheral interface (Jameco DP8255) |
| 1 | 7404 hex inverter |

- 1 74100 8-bit latch
- 1 seven-segment display DL707 (Radio Shack 276-053)
- 1 8-position DIP switch (Radio Shack 275-1301)
- 1 68- Ω resistor (blue-gray-black)
- 2 180- Ω resistors (brown-gray-brown)
- 10 1-k Ω resistors (brown-black-red)
- 2 pushbutton normally open (PBNO) switches
- 1 red LED
- 1 green LED

DISCUSSION: CONDITIONAL VERSUS UNCONDITIONAL I/O

All the interfacing problems considered to this point have involved *unconditional* I/O. The input or output device is assumed ready to accept or output data unconditionally. This technique is fine for fast peripherals or cases where the operator must be queried from the keyboard before the program continues.

But what about a device like an electronic printer? This electromechanical peripheral requires a finite length of time to print each character it is sent by the microcomputer. For example, an ASR-33 teletype prints at a rate of one character each 100 ms. Yet the microcomputer may be capable of outputting data at a rate as fast as one character every 10 μ s (0.01 ms). This is obviously much faster than the printer can handle. Data will be lost unless some means of *synchronizing* the microcomputer to the much slower printer is found.

One common method is to require the peripheral to supply a **BUSY/READY** flag that can be read by the microcomputer. In this way, data will still flow between the computer and peripheral, but conditionally. The condition being that the status flag must indicate **READY**. Figure 11-1 summarizes these two basic types of I/O.

Polled I/O

A logical question to ask at this point is: How does the computer know when this **READY** flag is set? In other words, how does it know when the peripheral is ready? The most common technique is to establish a *status port* that the microcomputer can poll periodically.

Figure 11-2 illustrates the bit assignments for a typical status port. This particular port monitors three **READY** flags, but as many as eight could be monitored from a single port. The microcomputer checks to see (poll) if any of its I/O devices have data or can accept data by testing each bit individu-

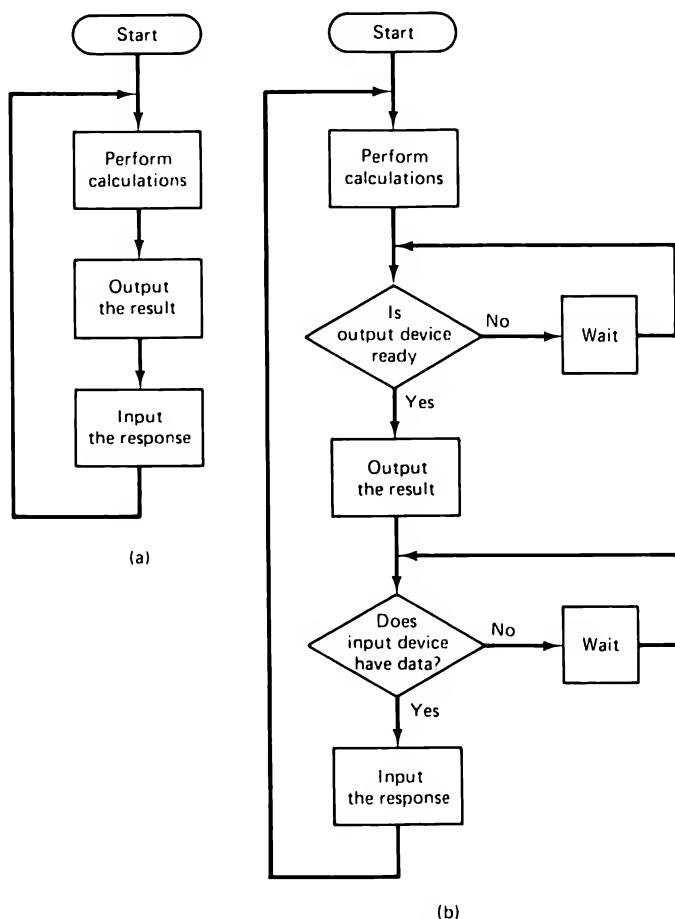


Figure 11-1 Comparing *conditional* and *unconditional* I/O. In (a) unconditional I/O is illustrated. The microcomputer assumes that the I/O device is always ready and inputs or outputs data accordingly. In (b), conditional I/O is shown. Now the computer is synchronized to the I/O device **READY** status.

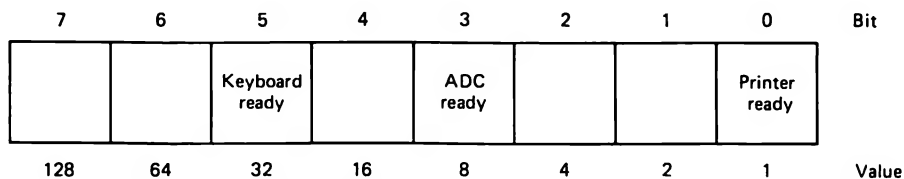


Figure 11-2 Bit assignments for a typical status port. One port can provide the **BUSY/READY** status for up to eight different peripherals.

ally. For example, an analog-to-digital converter (ADC) may take several milliseconds to convert its input voltage to an n -bit digital word. Finally, when ready, its **BUSY/READY** flag will be set, indicating that it is ready for new data. This condition can be tested by a BASIC program such as

```

10 GOSUB 500: IF B(3)<>1 THEN 10      :REM WAIT FOR READY
20 Y=PEEK(49393)
30 END
500 N=PEEK(49392)
510 FOR J=7 TO 0
520 B(J)=1
530 IF N-2^J<0 THEN B(J)=0: GOTO 550
540 N=N-2^J
550 NEXT J
560 RETURN

```

Line 10 waits for bit 3 of status port 49392 to go high (the converter is ready) and then inputs the digital word from data port 49393.

A more elaborate polling program could monitor all three flags and branch to different locations when a particular flag becomes ready. For example:

```

10 GOSUB 500: IF B(0)=1 THEN 100
20 IF B(3)=1 THEN 200
30 IF B(5)=1 THEN 300
40 GOTO 10

```

The standard decimal-to-binary subroutine is assumed in location 500, as in the previous example. Special routines to handle each I/O device are then placed in locations 100, 200, and 300.

A major *disadvantage* to polling is that the computer is continually tied up checking its I/O devices waiting for a **READY** flag to be set. The analogy has been made that polled I/O is similar to someone waiting for a phone call by continually picking up the receiver and asking if someone is there (rather than letting the bell ring—interrupting you from some other activity). Depending on the circumstances, this may be a rather inefficient use of the computer (not to mention a rather tiring way of waiting for a phone call!).

Interrupt-Driven I/O

The second common method of interfacing to conditional I/O devices is to use an *interrupt* technique. As explained in the phone call analogy, it would be more efficient to allow the computer to perform some other job, and only when the I/O device is ready, stop and service it. This is the essence of interrupt-driven I/O.

Most microprocessors have some form of interrupt capability. You may think of this as a hardwired line directly into the “heart” of the microprocessor. When this line is activated (for example, pulled low), the microprocessor will save the contents of the various registers it is working with in RAM memory (the “stack”) and then branch to a special memory location to service the interrupting device.

Several advantages of interrupt-driven I/O should come to mind. First, the microcomputer need only service the I/O device when that device is ready; the rest of the time the computer can be working on some other task. With *vectored* interrupts or *multilevel* interrupts, the memory location of the I/O device service routine is established at the time of the interrupt. This means there is no need for a GOTO type of statement to direct the computer to the correct routine.

Because of this, the handling of I/O seems almost “invisible” to the user. In fact, using this technique, the computer can appear to be doing two different jobs at one time. For example, a clock circuit might generate an interrupt once each second. The computer could service this interrupt (the background) by adding one to a count and display the time of day on the CRT screen. This might require 200 to 300 μ s, but leave 999,700 μ s before the next interrupt occurs. During this time the computer can work on its main task (the foreground). To the user, the computer appears to be doing two tasks simultaneously.

Another advantage to interrupts is that a *priority* structure can be established. If conflicts occur because of multiple interrupts, the priority structure will see that the most important device is serviced first. For that matter, the interrupts can be turned off (disabled) when the microcomputer does not want to be interrupted (wouldn't you like to be able to do that!).

The main disadvantage to interrupts is the increased software complexity. The computer can actually become interrupt bound, continually servicing interrupts without being able to return to its main task. Debugging errors in this type of software can be extremely difficult.

Generally, the servicing of interrupts must be done with an *assembly language* program. Interrupting a BASIC program can have disastrous results. When an interrupt occurs the current machine instruction is completed first before branching to the service routine. But with BASIC one program statement may involve many machine instructions. There is no guarantee that when you return from the interrupt service routine the many variables required by BASIC will all be unchanged.

Which Scheme Should I Use?

Despite the advantages of interrupt-driven I/O, for most small BASIC-oriented computer systems such as the Apple, *polling* may be the best choice.

Often in such systems the computer has nothing else to do while waiting for its I/O anyway, and the efficiency may be academic. For this type of computer the interrupt capability is best designed into the overall operating system ROMs.

Using the 8255 for Conditional I/O

The 8255 programmable peripheral interface was first introduced in Experiment 5. Since then we have used it in many of the experiments but always programmed it to operate in *mode 0*. In this experiment we are demonstrating conditional I/O and will use the 8255 in *mode 1*. Intel refers to this mode as *strobed input/output*.

Figure 11-3 illustrates the four combinations possible for mode 1 operation. These correspond to the four different combinations of ports A and B as input and output ports. Notice that port C is a *status* port and not a data port for this mode. As such, it provides strobe and acknowledge signals (sometimes referred to as “*handshaking*” signals) for the two data ports.

Figure 11-4 indicates the names and descriptions of these handshaking signals. For the case of an input port, the input device first pulls the strobe line (\overline{STB}) low, causing data to be gated immediately into the 8255 (but not the microcomputer). The 8255 *acknowledges* this data transfer by causing its IBF (input buffer full) line to go high. The microcomputer now has the choice of using a polled or an interrupt-driven technique to determine that data is ready and then input this data. If polling is used, the IBF line of port C (PC1 or PC5) must be tested and data subsequently read from port A or B as appropriate. This read operation also resets the IBF flag.

If interrupts are used, the INTR output of the 8255 must be connected to the interrupt input of the microprocessor. Branching to the proper service routine will now occur “automatically.” Notice that the interrupts must first be enabled by a bit set/reset operation in order for INTR to be active.

An output port is handled in a similar fashion. In this case, the microcomputer outputs data to the appropriate 8255 port, causing its \overline{OBF} (output buffer full) line to go low. This line should be monitored by the output device to determine when data is ready. The output device then *acknowledges* receipt of the data by driving the 8255 \overline{ACK} (acknowledge) line low.

Again the microcomputer may use polling or interrupts to interface this port. If polling is used, the \overline{ACK} signal of port C (PC2 or PC6) is tested and new data is output only when this line is low, indicating that the previous data has been accepted by the output device. The interrupt technique uses the INTR line to determine this same information. Again the INTR line must be high to enable interrupts.

Figure 11-5 will help you determine the proper control word to be used, depending on the port configuration desired. Notice that when configuring

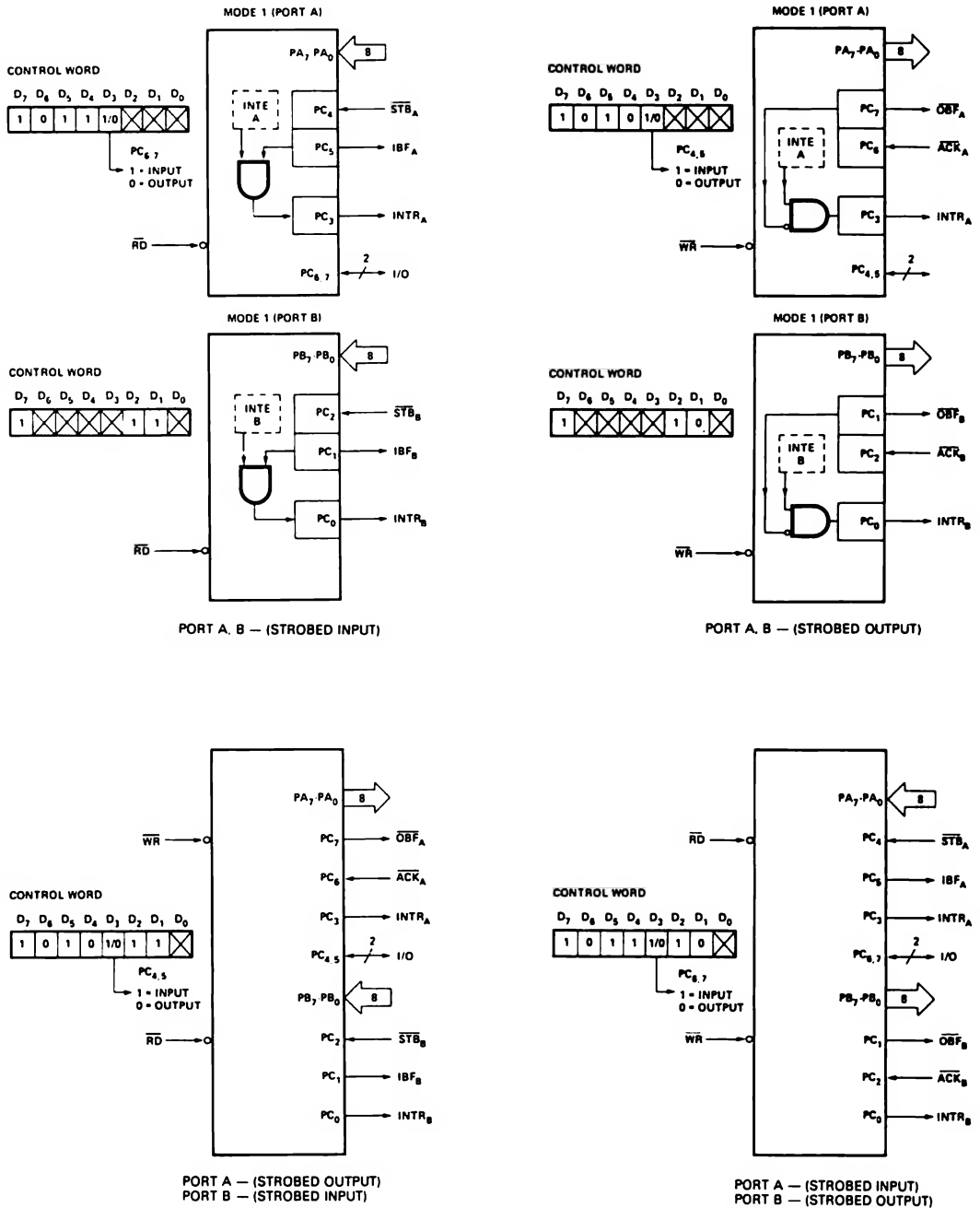


Figure 11-3 The four possible I/O configurations for ports A and B when the 8255 is programmed for mode 1 operation. Port C becomes a status port in this mode. (Courtesy of Intel Corporation.)

Input Port		
Signal	Direction	Description
IBF	OUT	A 1 on this output indicates that the data has been loaded into the input latch; in essence an acknowledgment. IBF is set by the falling edge of the \overline{STB} input and is reset by the rising edge of the \overline{RD} input.
\overline{STB}	IN	A 0 on this input loads data into the input latch.
INTR	OUT	A 1 on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the rising edge of \overline{STB} if IBF is a 1 and INTE is a 1. It is reset by the falling edge of \overline{RD} . This procedure allows an input device to request service from the CPU by simply strobing its data into the port. INTE A is controlled by bit set/reset of PC4 and INTE B by bit set/reset of PC2.
Output Port		
Signal	Direction	Description
\overline{OBF}	OUT	The \overline{OBF} output will go low to indicate that the CPU has written data out to the specified port. The \overline{OBF} flip-flop will be set by the rising edge of the \overline{WR} input and reset by the falling edge of the ACK input signal.
ACK	IN	A 0 on this input informs the 8255 that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.
INTR	OUT	A 1 on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set by the rising edge of ACK if \overline{OBF} is a 0 and INTE is a 1. It is reset by the falling edge of \overline{WR} . INTE A is controlled by bit set/reset of PC6 and INTE B by bit set/reset of PC2.

Figure 11-4 Port C of the 8255 supplies several “handshaking” signals when programmed for mode 1 operation.

the 8255, two lines of port C are always available for use as input or output bits independent of the other 6 bits.

Example 11-1

Determine the control word to be used to cause port A to be an output, port B an input, and PC4 and PC5 to be outputs.

Solution Referring to Fig. 11-5, the following binary pattern is found:

$$1010011X = 166 \text{ or } 167$$

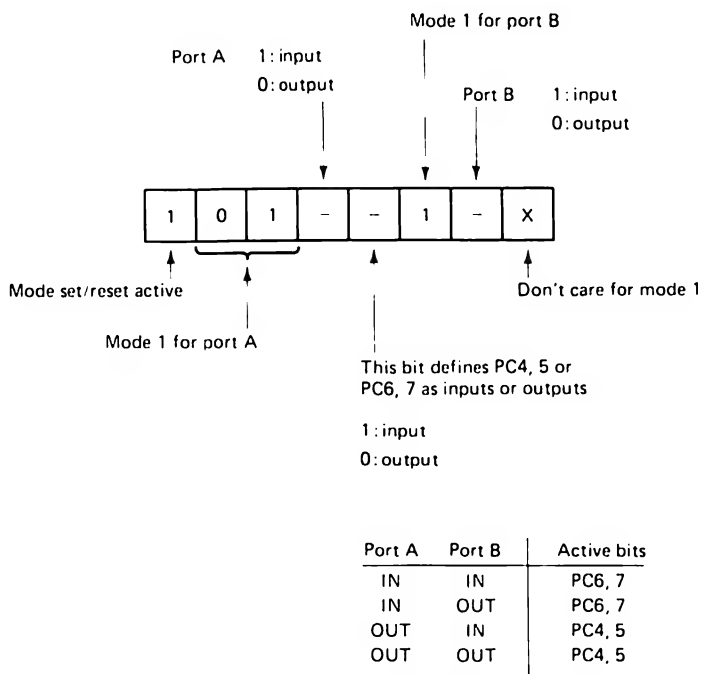


Figure 11-5 Mode 1 control word format.

PROCEDURE

Step 1. Refer to Fig. 11-6 and wire the 8255 on your breadboard. Now add the port A DIP switch, the IBF monitor at PC5, and the strobe input at PC4. The remainder of the circuit will be wired later.

Question 11-1. Write a program that configures the PPI in mode 0 and tests the DIP switch connections.

Question 11-2. What control word is needed to configure the PPI for *mode 1* with port A an input port and port B an output port and PC6 and PC7 inputs?

Step 2. Load and run the following program. With the program running, set a number up on the DIP switch and depress and release S1. Try this several times for different numbers on the switch.

```

10 HOME
20 POKE 49395,188      :REM PPI MODE 1, A=INPUT B=OUTPUT
30 PRINT PEEK(49392)   :REM READ SWITCH
40 GOTO 30

```

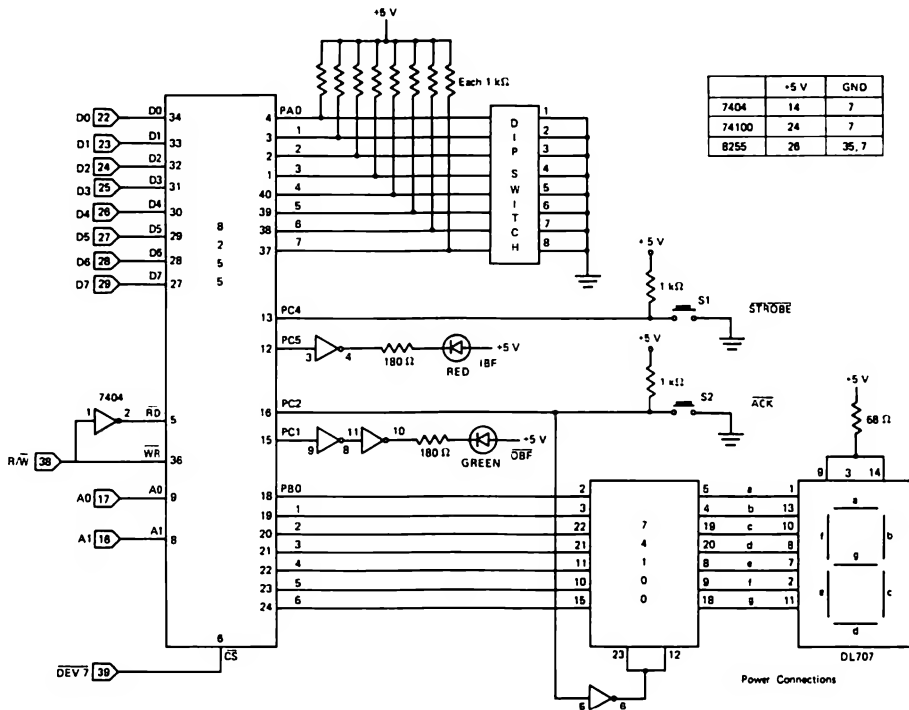


Figure 11-6 Circuit to illustrate handshaking logic with the 8255.

Question 11-3. Unless the strobe input is pulsed low, the computer output to the CRT screen remains unchanged. Explain.

Step 3. Change the program in step 2 by adding new lines 30 to 70 and the subroutine beginning at line 500 as follows:

```

30 GOSUB 500: IF B(5)<>1 THEN 30      :REM WAIT FOR IBF
40 FOR J=1 TO 500: NEXT J             :REM DELAY TO SEE IBF
50 VTAB(12)
60 PRINT "THE INPUT VALUE IS: ";PEEK(49392)
70 GOTO 30
500 N=PEEK(49394)                     :REM TEST PORT C
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF N-2^J<0 THEN B(J)=0: GOTO 550
540 N=N-2^J
550 NEXT J
560 RETURN

```

Question 11-4. Notice the **IBF** LED as the program in step 3 is run. What turns ON this LED? What turns it OFF?

Note. The port A interface just constructed provides full handshaking with the IBF, acknowledge signal, and the strobe input. Also note that the 8255 acts as a *buffer* by holding the input data until the microcomputer is ready to read it.

Step 4. Wire the 74100, seven-segment display, $\overline{\text{OBF}}$ LED monitor, and S2 acknowledge switch as shown in Fig. 11-6. Now load and run the following program with S2 depressed. You should observe the numeral 5 on the seven-segment display.

```
10 HOME
20 POKE 49395,153      :REM INIT PPI FOR MODE 0, B=OUTPUT
30 POKE 49393,18       :REM MAKE SEGMENTS b AND e HIGH (OFF),
                       :ALL OTHERS LOW (ON)
40 END
```

Question 11-5. Change the program in step 4 so that the display appears to *count* at a slow rate from 0 through 9. You should then be able to appreciate how software can take the place of a conventional seven-segment decoder.

Step 5. The following program tests the port B interface using mode 1. Load and run this program. Push S2 and observe the $\overline{\text{OBF}}$ LED and display.

```
10 HOME
20 POKE 49395,188      :REM PPI MODE 1
30 INPUT "WHAT NUMBER ";N
40 POKE 49393,N         :REM OUTPUT DATA
50 GOSUB 500: IF B(2)=1 THEN 50 :REM WAIT FOR ACK
60 GOTO 30
500 N=PEEK(49394)       :REM TEST PORT C
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF N-2^J<0 THEN B(J)=0: GOTO 550
540 N=N-2^J
550 NEXT J
560 RETURN
```

Question 11-6. Notice the $\overline{\text{OBF}}$ LED as the program in step 5 is run. What turns ON this LED? What turns it OFF?

Step 6. Write a program to demonstrate the full handshaking concept from inputting of data from the DIP switch to outputting that data back to the display at port B. Figure 11-7 is a flowchart of the process. A solution is provided at the end of this experiment.

Step 7. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

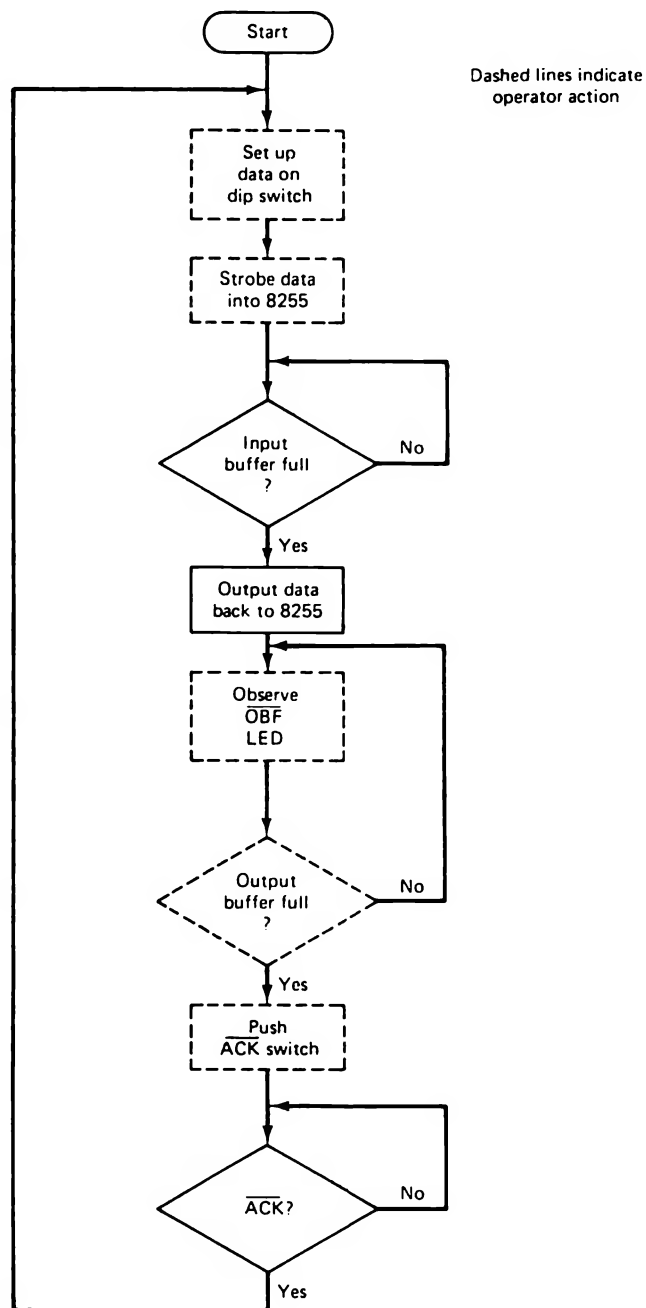


Figure 11-7 Flowchart of the full handshaking program to be written for step 6.

 SOLUTIONS TO QUESTIONS

11-1.

```

10 HOME
20 POKE 49395,153      :REM INIT PPI MODE 0 A=INPUT, B=OUTPUT
30 PRINT PEEK(49392)
40 GOTO 30

```

Note. This program also serves as a binary to decimal converter.

11-2. 1011110X = 188 or 189.

11-3. The data is not latched by the 8255 until the strobe input is pulsed low. Until this occurs, port 49392 contains the “old” data.

11-4. Closing S1 strobes the data into the 8255, turning the IBF LED ON. The PEEK(49392) operation resets the IBF line.

11-5.

```

10 HOME
20 FOR J=0 TO 9: READ N(J): NEXT J  :REM N(J) HOLDS THE
                                     SEVEN-SEGMENT PATTERNS
30 POKE 49395,153                  :REM INIT PPI
40 FOR J=0 TO 9
50 POKE 49393,N(J)
60 FOR I=1 TO 300: NEXT I          :REM SLOW COUNT
70 NEXT J
80 GOTO 40
90 DATA 64,121,36,48,25,18,3,120,0,24 :REM 0-9 PATTERNS

```

11-6. An output write to port 49393 turns on the $\overline{\text{OBF}}$ LED. It is reset by the acknowledge pulse produced by closing S2. This also clocks the data into the 74100 and seven-segment display.

Step 6 (solution).

```

10 HOME
20 POKE 49395,188              :REM INIT PPI
30 GOSUB 500: IF B(5)<>1 THEN 30 :REM WAIT FOR IBF
40 FOR J=1 TO 500: NEXT J      :REM TIME DELAY
50 Y=PEEK(49392)               :REM INPUT THE DATA
60 POKE 49393,Y                :REM NOW OUTPUT THE DATA
70 GOSUB 500: IF B(2)=1 THEN 70 :REM WAIT FOR ACK
80 GOTO 30
500 N=PEEK(49394)              :REM SAMPLE PORT C
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF N-2^J<0 THEN B(J)=0: GOTO 550
540 N=N-2^J
550 NEXT J
560 RETURN

```

EXPERIMENT 12

SERIAL INTERFACING

OVERVIEW

In this experiment you will wire the Motorola MC6850 *asynchronous communications interface adapter* (ACIA) to the address and data buses of the Apple. The 6850 will be wired to transmit to itself and a BASIC program will be used to display the ACIA status bits in real time.

OBJECTIVES

The key points to be learned from this experiment are:

1. The ACIA receives and transmits serial data at a baud rate that is typically $\frac{1}{16}$ of the clock rate.
2. The ACIA is programmed "automatically" to insert start bits, stop bits, parity bits, and to select the number of data bits per transmitted or received word.
3. A typical ACIA interface utilizes status bits to indicate when a word is ready to be received or transmitted.
4. Most computer terminals use a 7-bit data word and the ASCII code for letters, numerals, and punctuation symbols.
5. The 6850 ACIA provides TTL input and output voltages, but these must be converted to RS-232C levels for terminals and printers or converted by a modem to two different audio frequencies for transmission over the telephone lines.

PARTS LIST

- 1 6850 asynchronous communications interface adapter (Jameco MC6850)
- 1 7404 hex inverter
- 1 LM555 universal timer (Radio Shack 276-1723)
- 1 1-k Ω resistor (brown-black-red)
- 1 180- Ω resistor (brown-gray-brown)
- 1 LED
- 1 50-k Ω potentiometer
- 1 0.01- μ F capacitor
- 1 22- μ F capacitor

DISCUSSION: SERIAL CONCEPTS

The microcomputer is inherently a *parallel* machine. It communicates with its memory and I/O ports via eight parallel data bus lines. For this reason, all the interfaces we have constructed to this point have been parallel and dealt with these eight data lines.

Because of this, a serial I/O port requires reorganizing the data path to a single line through which the 8 data bits must pass in single file. Figure 12-1 compares typical serial and parallel computer ports (in this case to a printer).

Both ports are similar in that normal address decoding and control logic are needed to place the port at a particular memory or I/O address. In this respect the microcomputer cannot tell the difference between the two ports. However, what the two ports do with their data once it is received is quite different.

As shown in Fig. 12-1, the parallel port passes the eight lines on to the printer and includes two control lines (**BUSY/READY** and **STROBE**) and a common ground, for a total of 11 connections.

The serial port converts the data to single bits that are transmitted one at a time to the printer using the **DATA OUT** line. If the printer wants to reply, it may do so in a similar fashion using the **DATA IN** line. Finally, a common ground establishes the third connection.

The advantage of the serial technique is its *simplicity* and *low cost*—only a three-conductor cable is needed. The advantage of the parallel technique is its *high speed*, as all bits arrive at once.

The choice of which method to use usually depends on the distance between the computer and peripheral (lengthy multiconductor cables are very expensive) and the speed of transmission required. Often the peripheral is a human interface (printer or video terminal) and extreme high speed is not critical. In these cases the economy of serial transmission is chosen.

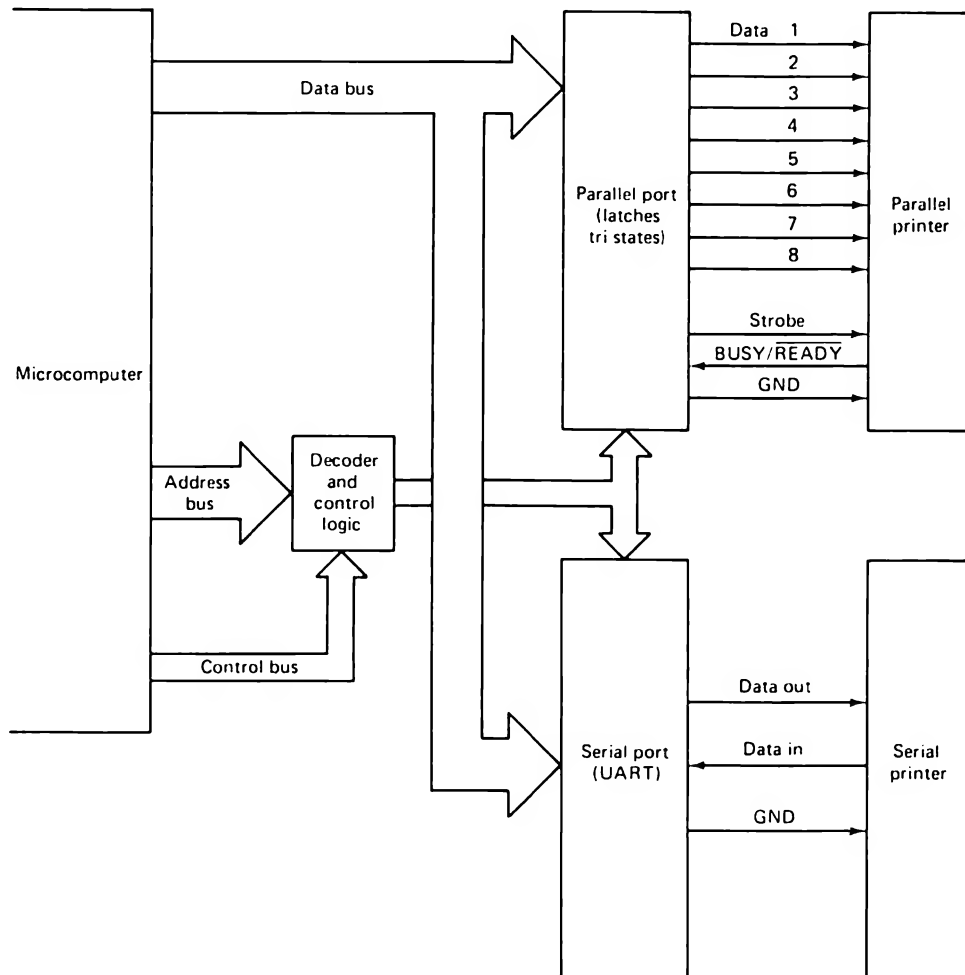


Figure 12-1 Comparing serial and parallel computer ports. The parallel port is much faster but also more complex than the serial port.

Another advantage of serial interfacing is that it may be used with a *modem* (modulator-demodulator) to pass data over the telephone lines for thousands of miles. There is more detail on this technique in the last section of this "Discussion."

Baud Rate

The rate at which the serial data bits are transmitted is referred to as the *baud rate*. A typical baud rate for a computer terminal is 1200. This means

that 1200 bits are transmitted each second. Although you might think that this would represent 150 bytes per second ($1200/8$), in practice, extra bits are sent along with the data bits, so that one character (or byte) may actually be as many as 12 bits long. This reduces the character rate to 100 characters per second in this example. But the baud rate remains 1200.

Several factors affect the choice of baud rate for a given interface. Because a printer is a mechanical device, we must be careful not to send it characters faster than it can type them out or data will be lost. For example, the ASR-33 teletype runs at 110 baud. At this rate it can only accept data at a rate of 10 characters per second.

When long cables (>1000 ft) are used, the internal capacitance of the cable limits maximum speed. Baud rates as high as 19,200 can be used with CRT terminals and short (<30 ft) cables.

ACIAs and UARTs

When comparing the serial and parallel ports in Fig. 12-1, it was stated that the serial port is less complex and more economical than the parallel port. However, you might question this since the serial port must convert the parallel data from the computer to the serial format. Fortunately, the IC manufacturers have recognized the importance of a circuit capable of doing this conversion, and have developed a special serial communications chip called the *universal asynchronous receiver-transmitter*, or UART for short. Motorola prefers the acronym ACIA for *asynchronous communications interface adapter*. For the purposes of this discussion the terms "UART" and "ACIA" are interchangeable.

The UART contains a separate and independent transmitter and receiver of serial data in one IC package. All that is needed is a clock circuit to establish the baud rate. Usually, this clock runs at *16 times* the actual baud rate. For example, at 1200 baud the actual clock rate is $16 \times 1200 = 19.2$ kHz. Figure 12-2 illustrates the key components in a typical UART.

The transmitter section receives 8 bits of data from the computer and begins transmitting the data when the *strobe* pulse occurs. The control section determines how many bits each transmitted word will have, the number of stop bits, and the choice of parity. The transmitter buffer empty flag (TBE) indicates if the transmitter is ready for a new data word.

Similarly, the receiver data ready (DR) flag indicates that a serial character has been received and is available to be read. Once the data has been read, the data ready flag should be reset by applying a *reset data ready* pulse as shown in Fig. 12-2. The control section is shared by the receiver and transmitter, so the word configuration must be the same for both.

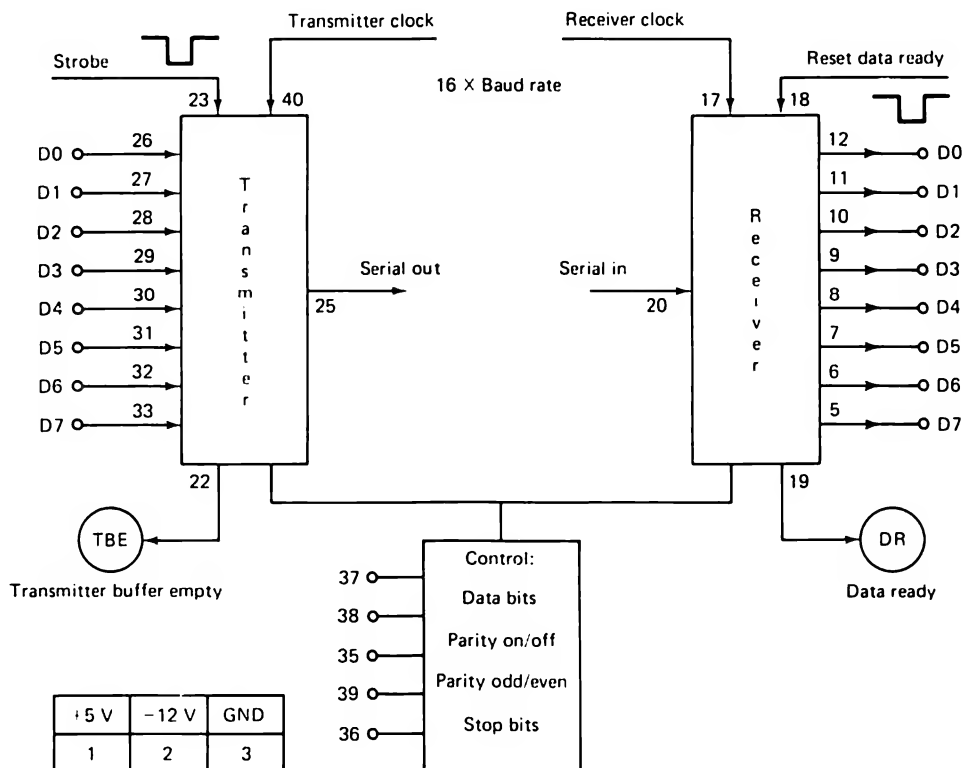


Figure 12-2 Block diagram of a typical UART. A separate receiver and transmitter are provided. Hardwiring control pins high or low determines the number of data bits per word, stop bits, and parity. The pin numbers shown are for the General Instrument AY-5-1013 UART.

Data Word Format

It was mentioned earlier that several extra bits are transmitted along with the data. These are the *start bit*, always a 0; one or two *stop bits*, each always a 1; and an optional *parity bit*. The parity bit may be chosen to be *odd* or *even*. This simply means that the total number of 1's in the word (including the parity bit) will be even or odd.

The purpose of the start and stop bits is to frame the data for synchronization. The receiver knows that whenever the data input line drops low (a start bit), data is about to be transmitted. The stop bit allows the receiver some time to get back in "sync" before the next character is sent. This is necessary because the receiver and transmitter clocks are usually located at different sites and therefore are not at the exact same frequency.

The purpose of the parity bit is to detect single bit errors in transmission. If a single 1 or 0 is lost, the received word will have the “wrong” parity. This will set a parity error flag in the UART and the receiver may then request a retransmission.

Example 12-1

Draw waveforms to scale for the output of a UART running at 1200 baud and transmitting the byte: 01010101. Assume odd parity and two stop bits.

Solution Refer to Fig. 12-3. Each character begins with a start bit and ends with the 2 stop bits. The 8 data bits are transmitted least significant bit first. The parity bit follows the data bits and is a 1 to maintain an odd number of 1's in the word (five in this case). Finally, two stop bits end the word.

At 1200 baud, one bit lasts for $1/1200 \text{ s} = 833 \mu\text{s}$. It takes $12/1200 \text{ s} = 10 \text{ ms}$ to transmit the entire word. This corresponds to 100 characters per second.

The UART shown in Fig. 12-2 is typical of the *General Instrument AY-5-1013*. This type of UART allows most of the control functions to be hardwired high or low and simplifies software control.

Another type of UART is represented by the *Intel 8251* and *Motorola MC6850*. These devices are similar to the *Intel 8255* programmable peripheral interface because they are controlled totally from software. In both of these devices there is one data path, the bidirectional 8-bit data bus. Received and transmitted data is sent over these lines along with control

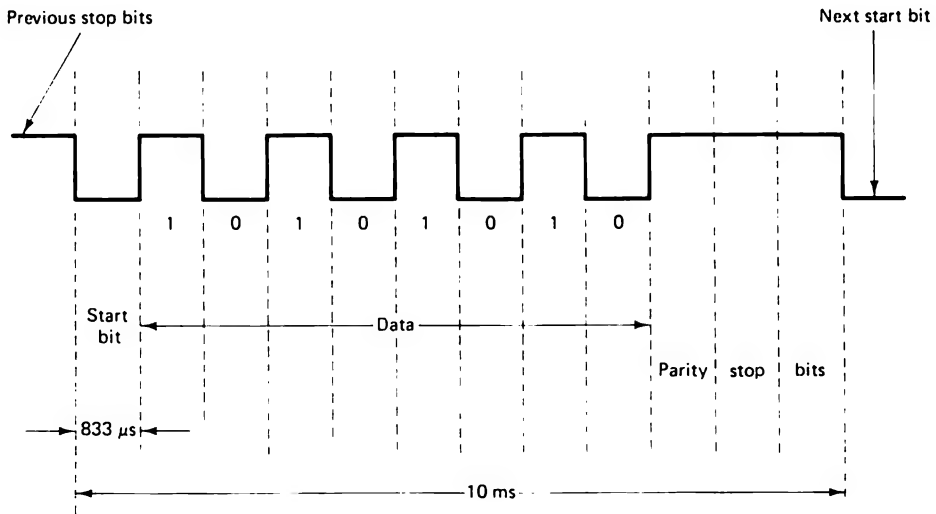


Figure 12-3 Output waveforms for a UART transmitting the byte 01010101 with odd parity and 2 stop bits. The baud rate is 1200.

information establishing parity, number of stop bits, and number of data bits per word. In the next section we will study the MC6850 in detail and show an interface to the Apple computer.

Interfacing the 6850 ACIA to the Apple

Figure 12-4 illustrates the electrical connections required to interface the 6850 to the Apple. All communication with the ACIA is via the 8-bit bidirectional data bus wired to pins 15 through 22. To the Apple, the 6850 appears to be two sequential memory locations corresponding to RS=0 and RS=1. The specific memory address is determined by the three chip select inputs, CS0, CS1, and $\overline{\text{CS2}}$. When these inputs equal 1, 1, 0, respectively, the 6850 is selected. The register select input (RS) and $\text{R}/\overline{\text{W}}$ input then determine the specific internal register selected as illustrated in Table 12-1.

Studying Table 12-1, when RS=0 (that is, A0=0 in Fig. 12-4) the output *control* register or input *status* register is selected. The $\text{R}/\overline{\text{W}}$ line controls which of these two will be accessed. These registers allow the microprocessor to set the *protocol* of transmitted words (bits per word, parity, stop bits, and clock rate) and to monitor the status of the transmitter and receiver buffers.

When RS=1 the *data* registers are selected. In this case a memory write operation will cause the output byte to be converted to serial and transmitted bit by bit at the specified clock rate. A memory read operation will input the word currently stored in the receiver buffer (that is, the word most recently received and converted to parallel).

Example 12-2

Indicate the four BASIC commands required to access the Apple-ACIA interface in Fig. 12-4 and describe the function of each.

Solution Examining the chip select inputs, we see that this interface is *partially* decoded because the A1 address line is not used. Because the $\overline{\text{DEV7}}$ device select signal from the Apple is connected to CS2, the range of possible addresses is 49392 through 49407 (refer to Experiment 2 if this is not clear to you). The low 4 bits of the address correspond to

A3	A2	A1	A0	
1	1	X	0	Control functions
1	1	X	1	Data functions

The offset for the control port is 12 choosing the don't care to be 0 (A3 A2 A1 A0 = 1100) and the data port offset is 13 (A3 A2 A1 A0 = 1101). The

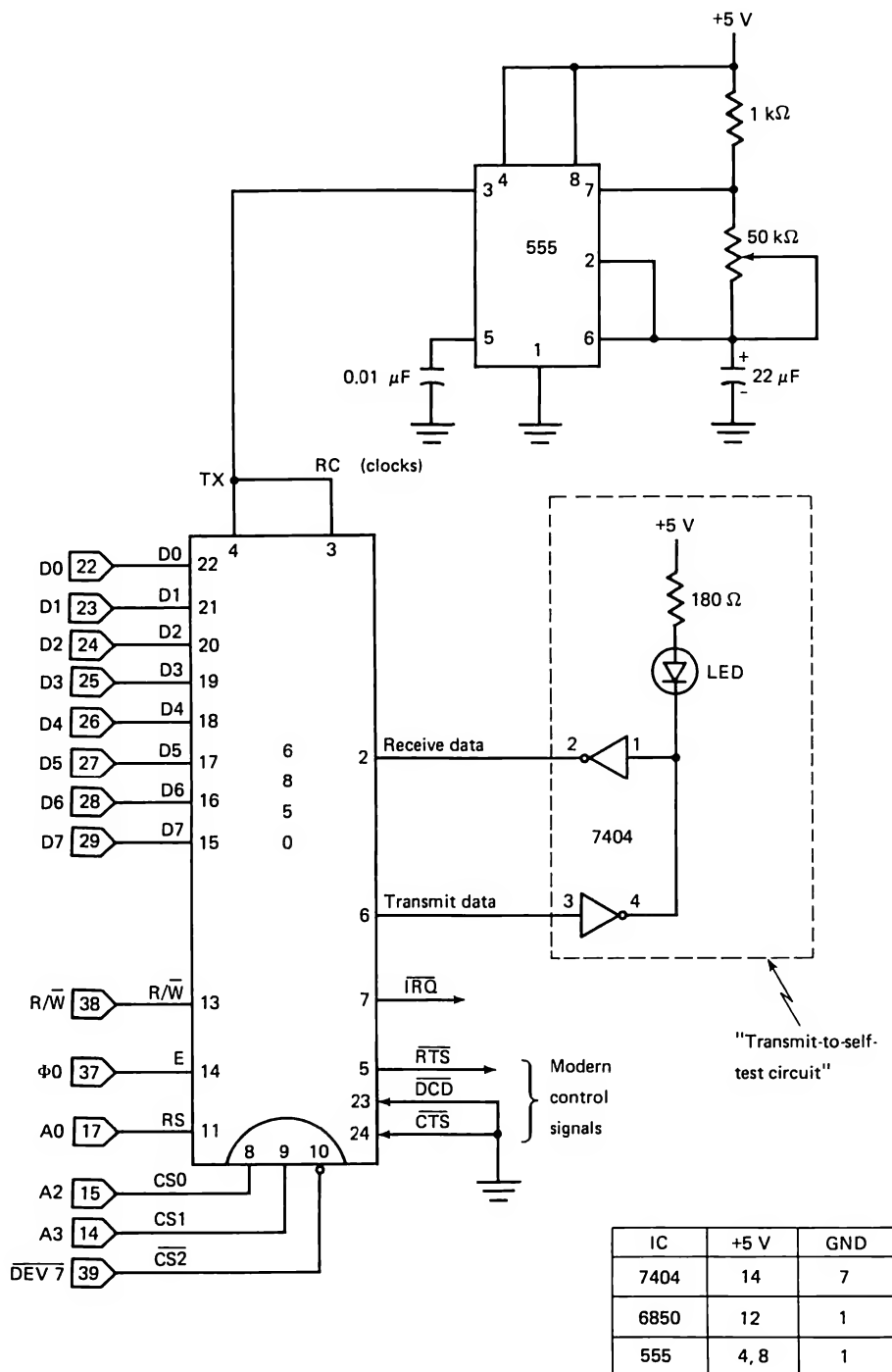


Figure 12-4 Interfacing the MC6850 ACIA to the Apple. This circuit occupies two memory locations at addresses 49404 and 49405.

TABLE 12-1 CONTROL FUNCTIONS OF THE 6850 ACIA

CS0	CS1	$\overline{\text{CS2}}$	RS	R/ $\overline{\text{W}}$	Function
1	1	0	0	0	Write to control register
1	1	0	0	1	Read from status register
1	1	0	1	0	Output data for transmission
1	1	0	1	1	Input received data

four commands and their functions are:

POKE 49404, Y	Y=OUTPUT CONTROL DATA
Y=PEEK(49404)	Y=STATUS INFORMATION
POKE 49405, Y	Y=OUTPUT DATA
Y=PEEK(49405)	Y=INPUT DATA

Note. Because address line A1 is not decoded, a second set of addresses will also respond to the PEEK and POKE commands. These addresses are 49406 (status and control port) and 49407 (data port).

In addition to the chip select signals and register select input, a *synchronization* signal called **E** must be connected to pin 14 of the ACIA. In Fig. 12-4 the $\Phi 0$ clock signal is used to provide this synchronization. Recall from Experiment 1 that only when $\Phi 0$ is high does the 6502 microprocessor use its data and address buses.

The clock or baud rate generator in Fig. 12-4 uses a 555 universal timer. Although this circuit is suitable for the purposes of this experiment, a more stable clock reference should be selected for a real-world application. The Motorola MC14411 *baud rate generator* is shown in Fig. 12-5. This chip simultaneously generates 14 different baud rates from 75 baud to 9600 baud using a 1.8432-MHz crystal. The crystal provides stability and accuracy, which is particularly important because the baud rates for the receiving and transmitting UARTs must typically match within 5%.

ACIA Control Register

Figure 12-6 illustrates the control designations for the 8 bits of the 6850 control register (address 49404 for the interface in Fig. 12-4). Before writing a specific pattern to this register, a *master reset* command should be given. For reasons that will become clear later in this “Discussion,” the RTS (request to send) output should be kept high during this reset to avoid a loss of data when communicating with a modem. The proper control word is then

$$\text{X10XXX11} = 67 \quad (\text{choosing X's} = 0)$$

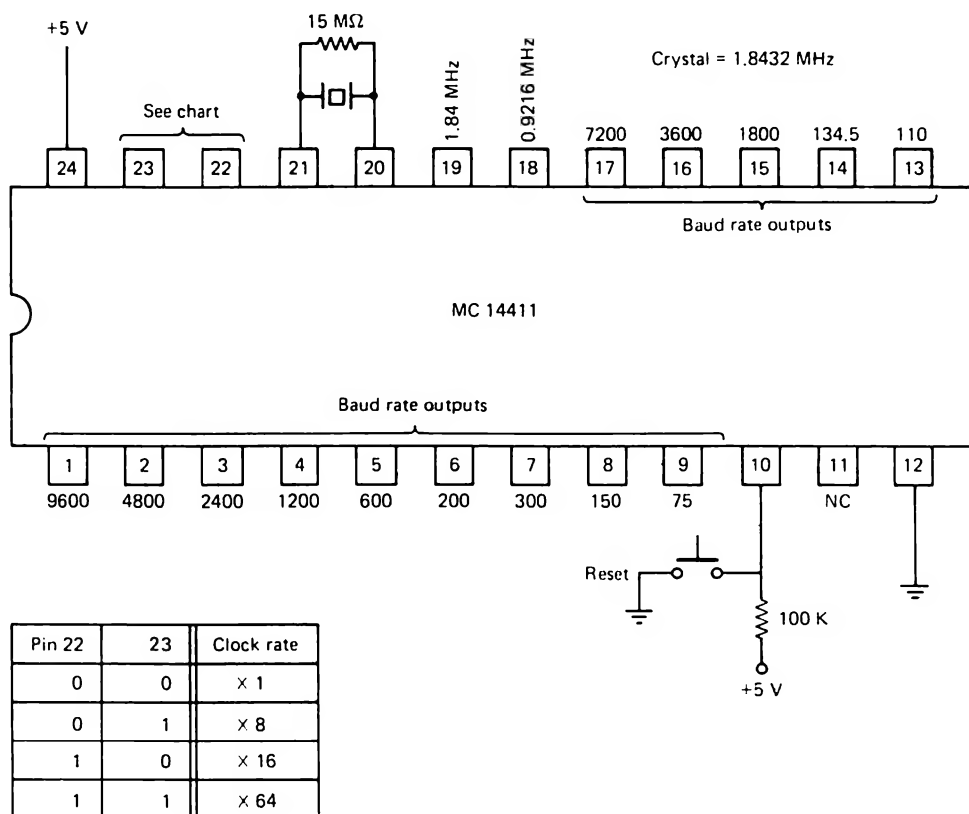


Figure 12-5 The MC14411 generates 14 different baud rates from its 1.8432-MHz crystal time base. Pins 22 and 23 select the output clock rate as 1X, 8X, 16X, or 64X the selected baud rate as required by the UART chosen.

Example 12-3

Determine the BASIC command sequence to be given to establish the following serial transmission protocol:

÷ 16 clock rate
 8 data bits
 no parity
 2 stop bits

Assume that the ACIA is used to communicate with a video terminal (not a modem) and that polling rather than interrupts will be used.

Solution The ÷16 clock rate requires $b1\ b0 = 01$. Similarly, the 8 data bits, no parity, and 2-stop-bit selection requires $b4\ b3\ b2 = 100$. Because communication is with a terminal, the modem control signal RTS is not needed. And because poll-

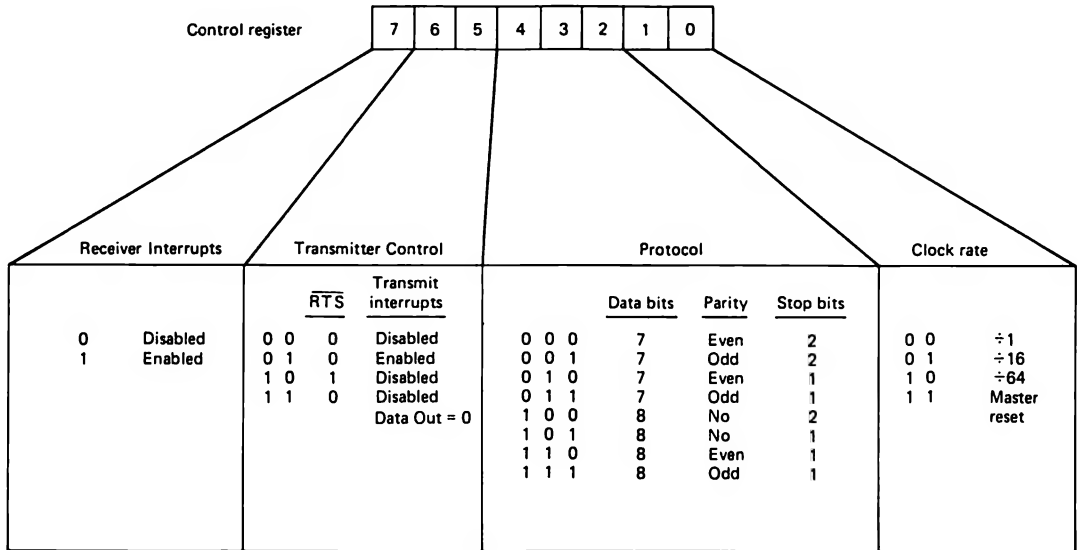


Figure 12-6 Port 49404 of the interface in Fig. 12-4 accesses the 6850 control register. The 8 bits of this port are used to specify the serial protocol desired, clock rate, and interrupt control logic.

ing will be used, the interrupt enables also will not matter. This means that b7 b6 b5 = XXX. Assembling the control word

$$\text{XXX10001} = 17 \quad (\text{choosing X's} = 0)$$

For the interface in Fig. 12-4 the initialization sequence would be

```
POKE 49404,67      :REM MASTER RESET
POKE 49404,17      :REM INITIALIZE
```

ACIA Status Register

Serial data transmission is of course slower than parallel transmission, requiring the microcomputer to synchronize itself to the character rate of the UART or ACIA. In the case of the 6850 ACIA, a status port is provided that can be polled to determine when the receiver has a new character to be read or the transmitter is ready to accept new data. Figure 12-7 illustrates the 8 status bits provided by the 6850. Two of these signals are particularly important for computer to ACIA synchronization.

Bit 0 of the status register is the receive data register full (**RDRF**) or data ready flag. This bit goes high when the full character has been received. It is cleared when the data register is read.

Bit 1 of the status register goes high when the transmitter data register

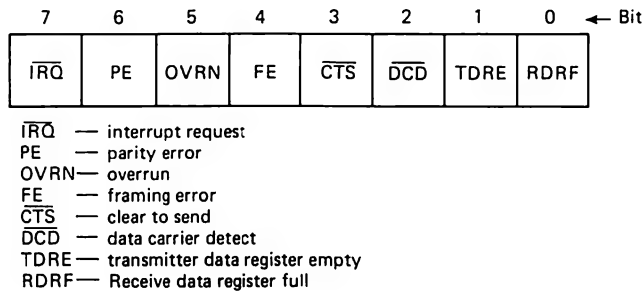


Figure 12-7 The command `Y=PEEK(49404)` will access the 6850 status register in Fig. 12-4. These 8 bits provide information about the transmitted and received data as explained in the text.

is empty (TDRE). This is called transmitter buffer empty (TBE) by some manufacturers. This bit can be tested by the microcomputer to determine when the ACIA is ready to transmit another character.

Three *error* conditions can be monitored at bits 4, 5, and 6. The *framing error* condition is set if the start and stop bits are not received at the proper time (that is, the ACIA is out of “sync”). The *overrun* condition occurs if the computer does not read the input data register before another character is received. When the received character’s parity does not match the parity specified (even or odd) by the control register, the parity error bit is set.

The 6850 may be interfaced using an interrupt-driven or a polled I/O technique (refer to Experiment 11 for more details on these two types of I/O). If interrupts are enabled—by choosing the appropriate bit pattern for the control register—the $\overline{\text{IRQ}}$ output at pin 7 will go low if bits 0, 1, 2, or 5 of the status register are set. In addition, bit 7 of the status register goes high whenever any interrupt occurs. This allows the 6850 to identify itself as the interrupting device.

Finally, bits 2 and 3 are modem status signals and will be discussed later under “Digital Communications.”

UART Software

Usually, the UART is controlled by a machine language program in ROM. Data is transmitted or received by simply sending or receiving characters to or from a special *driver routine*. In this experiment we are attempting to understand serial communications and will use BASIC to control the UART.

Figure 12-8 is a flowchart of a test program that utilizes the “transmit to self” test circuit in Fig. 12-4. Notice how the TDRE and RDRF status bits synchronize the microcomputer to the UART character rate.

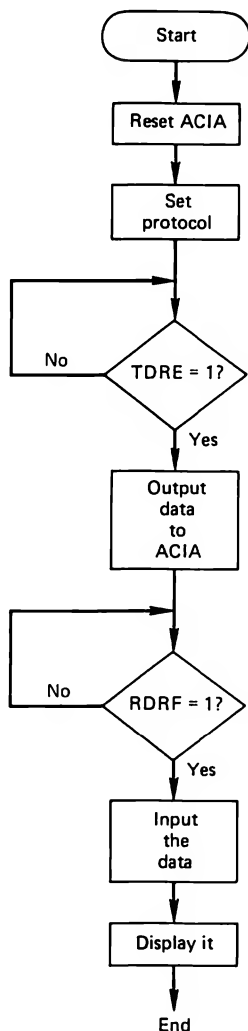


Figure 12-8 ACIA control flowchart for transmit to self test circuit.

Example 12-4

Write a BASIC program to transmit the 8-bit data word 01010101 (85_{10}) and then receive it using the “transmit to self” test circuit in Fig. 12-4. Use 2 stop bits and no parity.

Solution The program follows the flowchart of Fig. 12-8.

10	HOME	
20	POKE 49404,67	:REM MASTER RESET
30	POKE 49404,17	:REM SET PROTOCOL
40	GOSUB 500: IF B(1)<>1 THEN 40	:REM WAIT FOR TDRE
50	POKE 49405,85	:REM 85 IS THE DATA
60	GOSUB 500: IF B(0)<>1 THEN 60	:REM WAIT FOR RDRF

```
70 PRINT "THE DATA IS: ";PEEK(49405)
80 END
500 Y=PEEK(49404) :REM READ STATUS REGISTER
510 FOR J=7 TO 0 STEP -1
520 B(J)=1
530 IF Y-2^J<0 THEN B(J)=0: GOTO 550
540 Y=Y-2^J
550 NEXT J
560 RETURN
```

Digital Communications

Once your computer can transmit and receive serial data, an entire new world of digital communications is open to you. For example, with a serial port and a modem you may "call up" other computers over the telephone lines and participate in several of the *time-sharing networks* now available. Or call your stockbroker and get the latest quotations. Perhaps you need information on the undersea life of the *marine iguana* of the Galapagos Islands. The Dialog Information Retrieval Service (part of Lockheed Missile and Space Company, Inc.) has some 50 billion bytes of information available on line corresponding to about 40 million individual bibliographic abstracts and references!*

When using your computer to communicate with other computers (or terminals), certain standards must be followed by everyone for compatibility.

1. The *American Standard Code for Information Interchange* (ASCII). This is a 7-bit code that defines a particular binary pattern for all the letters of the alphabet, numerals, punctuation marks, and control characters. As long as the ASCII code is used by all manufacturers, when you type a question mark on your keyboard, it will also appear as a question mark on the receiving computer's CRT screen or printer. Table 12-2 is an ASCII code chart included for reference.

2. The *RS-232C* Serial Communications standard. This standard defines the voltage levels for logic 1's and 0's when communicating serially. You might be surprised to learn that this standard is not TTL compatible. A logic 1 is a voltage more negative than -3 V and a logic 0 must be more positive than +3 V. Typical levels used are ± 12 V. The reason for the large voltage levels is to provide greater noise immunity, particularly when long cable lengths are involved. The RS-232C standard goes beyond defining voltage levels. It also specifies a 25-conductor cable and connector—referred to as a standard DB-25 connector. Although signals are defined for all 25 pins, communications can take place using only three of those pins (pin 2—*transmit data*, pin 3—*receive data*, and pin 7—*signal ground*). Figure 12-9

*Stan Miastkowski, "Information Unlimited: The Dialog Information Retrieval System," *BYTE*, June 1981, pp. 88-108.

TABLE 12-2 THE 128 ASCII CHARACTERS AND THEIR DECIMAL EQUIVALENTS

DEC	ASCII	DEC	ASCII	DEC	ASCII
0	null	43	+	86	V
1	CONTROL	44	,	87	W
2		45	—	88	X
3		46	.	89	Y
4		47	/	90	Z
5		48	0	91	[
6		49	1	92	\
7		50	2	93]
8		51	3	94	^
9		52	4	95	_
10		53	5	96	@
11		54	6	97	a
12		55	7	98	b
13		56	8	99	c
14		57	9	100	d
15		58	:	101	e
16		59	;	102	f
17		60	<	103	g
18		61	=	104	h
19		62	>	105	i
20		63	?	106	j
21		64	@	107	k
22		65	A	108	l
23		66	B	109	m
24		67	C	110	n
25		68	D	111	o
26		69	E	112	p
27	ESC	70	F	113	q
28	FS	71	G	114	r
29	GS	72	H	115	s
30	RS	73	I	116	t
31	US	74	J	117	u
32	SP	75	K	118	v
33	!	76	L	119	w
34	"	77	M	120	x
35	#	78	N	121	y
36	\$	79	O	122	z
37	%	80	P	123	{
38	&	81	Q	124	
39	'	82	R	125	}
40	(83	S	126	~
41)	84	T	127	DEL
42	*	85	U		

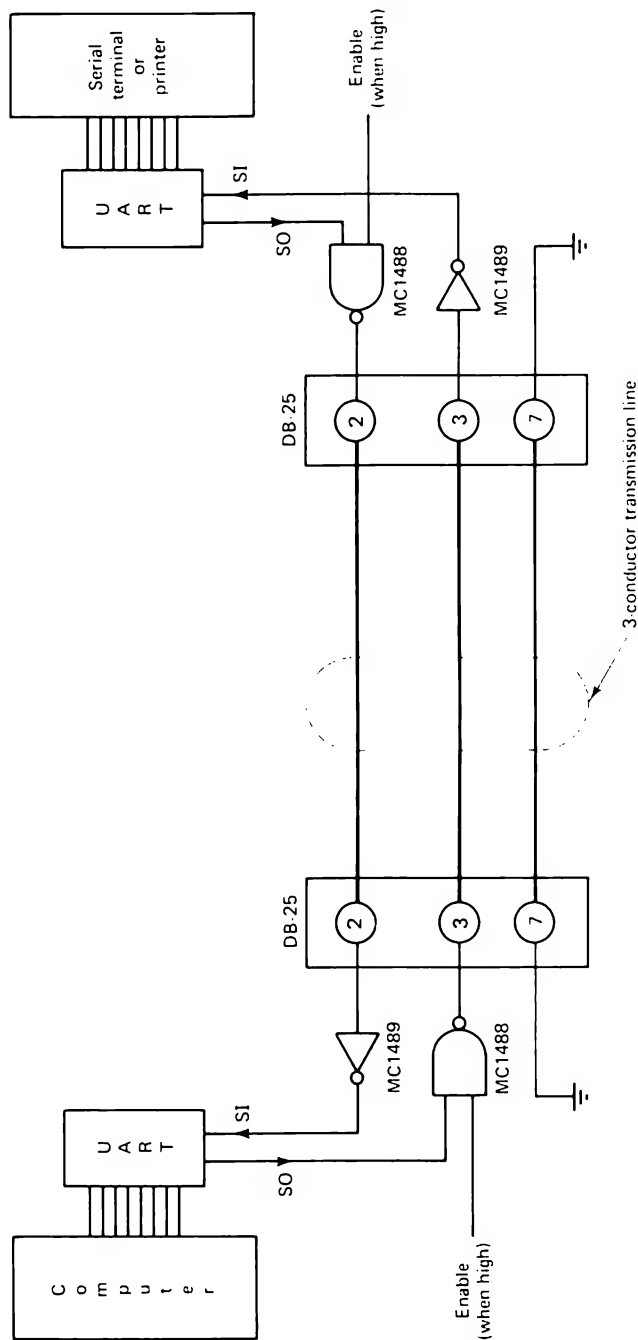


Figure 12.9 Typical RS-232C interface. Note that the **SERIAL IN** and **OUT** lines are reversed at each end of the interface and an MC1488 and 1489 are used to convert the UART's TTL levels to RS-232C levels. The pin numbers shown are for the standard DB-25 connector.

illustrates a typical RS-232C interface. Note the use of the MC1488 and MC1489 to convert the TTL levels to RS-232C levels.

3. *Modems.* A modem or modulator-demodulator converts the RS-232C voltage levels to two different audio tones. For example, an *originate* modem sends a 1270-Hz sine wave for a logic 1 and a 1070-Hz sine wave for a logic 0. Modems are used when communicating over long distances via the telephone lines. In situations like this the digital pulses would be hopelessly lost over the long line lengths involved, but the audio tones pass quite happily over this voice network. When using *full duplex* operation, transmission and reception at the same time, the receiving modem (called the *answer* modem) uses yet another set of frequencies, 2225 Hz for a 1 and 2025 Hz for a 0. When communicating with a modem the data rate is generally restricted to 300 baud or less.

The 6850 ACIA provides three modem control signals, called $\overline{\text{DCD}}$, data carrier detect; $\overline{\text{RTS}}$, request to send; and $\overline{\text{CTS}}$, clear to send. These signals are part of the RS-232C standard and correspond to pins 8, 4, and 5 on the DB-25 connector.

The typical sequence of events required to establish serial communications with a modem using the 6850 ACIA is:

1. Dial the phone number of the distant computer and wait to hear a high-pitched tone.
2. Connect the telephone to the modem. When the modem detects this carrier signal it will pull the $\overline{\text{DCD}}$ input to the ACIA low ($\overline{\text{DCD}} = 0$).
3. When $\overline{\text{DCD}}$ goes low, the serial transmit logic within the ACIA is enabled. The microcomputer may now "request to send" ($\overline{\text{RTS}} = 0$) by writing the appropriate bit pattern to the control register bits 5 and 6 of the ACIA.
4. If the computer at the other end of the line is ready, the modem replies with "clear to send" ($\overline{\text{CTS}} = 0$). With half duplex lines $\overline{\text{CTS}}$ is held high while the distant computer is transmitting.

Note. When using the 6850 ACIA to communicate with a terminal instead of a modem, the $\overline{\text{DCD}}$ and $\overline{\text{CTS}}$ inputs must be grounded for proper operation.

PROCEDURE

Step 1. Refer to Fig. 12-4 and wire the ACIA interface circuit to your Apple. Again use caution when handling the static-sensitive 6850 device.

Step 2. Study Example 12-4. Now adjust the 50k pot to minimum resistance (maximum frequency) and then load and run the program given in this example. You should observe the LED indicator to flash ON and OFF as the character is being transmitted. If all goes well, the message "THE DATA IS: 85" will appear on the screen.

Question 12-1. How many total clock pulses are required to transmit the character?

Question 12-2. The 555 clock frequency should be adjustable between approximately 1 and 100 Hz. Calculate the time required to transmit the character using the program in step 2 at 1- and 100-Hz clock rates.

Question 12-3. Calculate the *baud rate* for these two clock frequencies.

Step 3. Change the control code in line 30 from 17 to 18 and rerun the program. Have the 555 adjusted to its maximum frequency.

Question 12-4. Explain the effect produced by the new control code. What happens to the *baud rate* when the control code is changed to 18?

Step 4. We may gain a better understanding of the 8 status bits by monitoring them as a character is being transmitted. Load and run the following program. Run the program several times until you understand the sequence of events.

Note. It will be necessary to slow the 555 clock down so that BASIC does not miss any of the status bit changes.

```

10 HOME
20 INVERSE
30 PRINT "IRQ PE OR FE CTS DCD TDR RDR"
40 NORMAL
50 POKE 49404,67           :REM MASTER RESET
60 POKE 49404,17          :REM 8 DATA BITS, ÷16 CLOCK
70 Y=PEEK(49404)          :REM SAMPLE STATUS PORT
80 GOSUB 500:GOSUB 600     :REM UPDATE SCREEN
90 IF B(1)<>1 THEN 70      :REM WAIT FOR TDRE
100 POKE 49405,85:PRINT "LOAD BUFFER"
110 Y=PEEK(49404)         :REM SAMPLE STATUS PORT
120 GOSUB 500:GOSUB 600   :REM UPDATE SCREEN
130 IF B(0)<>1 THEN 110    :REM WAIT FOR RDRF
140 PRINT "THE CHARACTER IS: ";PEEK(49405)
150 END
500 FOR J=7 TO 0 STEP -1
510 B(J)=1
520 IF Y-2^J<0 THEN B(J)=0:GOTO 540
530 Y=Y-2^J
540 NEXT J
550 RETURN
600 FOR J=0 TO 7
610 HTAB(30-J*4)
620 PRINT B(J);
630 NEXT J
640 PRINT
650 RETURN

```

Question 12-5. Which bit must be set before the message "LOAD BUFFER" is printed? Which bit must be set before the character itself is printed?

Question 12-6. Normally, the ACIA is used to transmit 7-bit ASCII characters. Assuming odd parity and 1 stop bit, what changes should be made to the program in step 4? Try it.

Step 5. Modify the program from step 4 so that it prompts you for: clock rate ($\div 16$ or $\div 64$), even or odd parity (use 1 stop bit and 7 data bits), and the ASCII character to be sent. The program should then modify the control code and transmit and receive the character. A solution is provided at the end of this experiment.

Step 6. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

SOLUTIONS TO QUESTIONS

12-1. $11 \text{ bits/character} \times 16 \text{ clock pulses/bit} = 176 \text{ clock pulses.}$

12-2. At 1 Hz the total time is 176 s. At 100 Hz the total time is 1.76 s.

12-3. Baud rate = bits/second. At 1 Hz: $1 \text{ bit}/16 \text{ s} = 0.06 \text{ baud.}$ At 100 Hz: $1 \text{ bit}/0.16 \text{ s} = 6.3 \text{ baud.}$

12-4. When the control code = 18, 64 clock pulses are required for each bit. This slows the rate of transmission by a factor of 4.

12-5. bit 1, TDRE; bit 0, RDRF

12-6. Change line 60 to POKE 49404,13 and line 140 to

PRINT "THE CHARACTER IS: ";CHR\$(PEEK(49405))

Step 5 (solution). Add or change the following lines to the program given in step 4.

```

12 INPUT "(1) 1/16 OR (2) 1/64 CLOCK RATE ";C
13 IF C=1 THEN CC=1: GOTO 15      :REM CC=CONTROL CODE
14 CC=2
15 INPUT "(1) EVEN OR (2) ODD PARITY: ";P
16 IF P=1 THEN CC=CC+8: GOTO 18
17 CC=CC+12
18 INPUT "WHAT CHARACTER DO YOU WISH TO SEND ";C$
19 HOME
60 POKE 49404,CC
100 POKE 49405,ASC(C$): PRINT "LOAD BUFFER"
```

EXPERIMENT 13

— INTERFACING A PROGRAMMABLE SOUND GENERATOR —

OVERVIEW

In this experiment you will interface a General Instrument *programmable sound generator* to the Apple using an 8255 PPI chip. BASIC programs will be developed to test all internal registers of the device. A program that converts the Apple and keyboard into an electronic organ with memory is given.

OBJECTIVES

The key points to be learned from this experiment are:

1. The AY-3-8910 programmable sound generator is designed for computer control and does not require external resistors or capacitors to achieve the various sound effects.
2. The AY-3-8910 has two modes of operation: *fixed amplitude* and *envelope generator* control.
3. The data bus and address bus of the AY-3-8910 are *time multiplexed*, requiring the interface to follow a specific sequence when writing data to the chip.
4. In some cases the execution of BASIC commands is too slow for strobing I/O devices, and external hardware must be designed to develop proper strobe pulse widths.

PARTS LIST

- 1 AY-3-8910 programmable sound generator (Digital Research: Computers, P.O. Box 401565, Garland, TX 75040)

- 1 8255 programmable peripheral interface (Jameco DP8255)
- 1 4013 dual D flip-flop (CMOS)
- 1 4069 hex inverter (CMOS)
- 1 7400 quad NAND gate
- 1 7404 hex inverter
- 1 74121 one-shot
- 1 LM386 audio amplifier (Radio Shack 276-1731)
- 1 10-M Ω resistor (brown-black-blue)
- 1 300- Ω resistor (orange-black-brown)
- 2 4.7-k Ω resistors (yellow-violet-red)
- 1 1-k Ω resistor (brown-black-red)
- 1 470- Ω resistor (yellow-violet-brown)
- 1 20-pF capacitor
- 1 0.001- μ F capacitor
- 1 0.1- μ F capacitor
- 1 300-pF capacitor
- 1 100- μ F capacitor
- 2 10- μ F capacitors
- 1 3.579545-MHz crystal (Radio Shack 272-1310)
- 1 loudspeaker

DISCUSSION: ELECTRONIC SOUND GENERATORS

Computers have been used to generate music and various sound effects since their inception. Early microcomputers were programmed to execute specific time-delay loops. The cyclic pattern established on the address bus as these loops were executed radiated to nearby transistor radios and music could be created.

More recent techniques have involved turning on and off oscillator circuits pretuned to produce specific notes. This method is illustrated in Fig. 13-1. A computer output port is used to activate the particular notes desired. Chips such as the Mostek MK5024ON *top-octave generator* have simplified this design. This IC, illustrated in Fig. 13-2, incorporates all the circuitry needed to produce one full octave on the equal-tempered scale in a single IC package.

The most recent developments have been to produce ICs capable of *complex sound* generation. This includes *tone*, *noise*, and control of the envelope *attack* and *decay* times. With these ICs it is possible to simulate the sound of a train whistle, police siren, gunshot, explosion, and thousands of other sound effects.

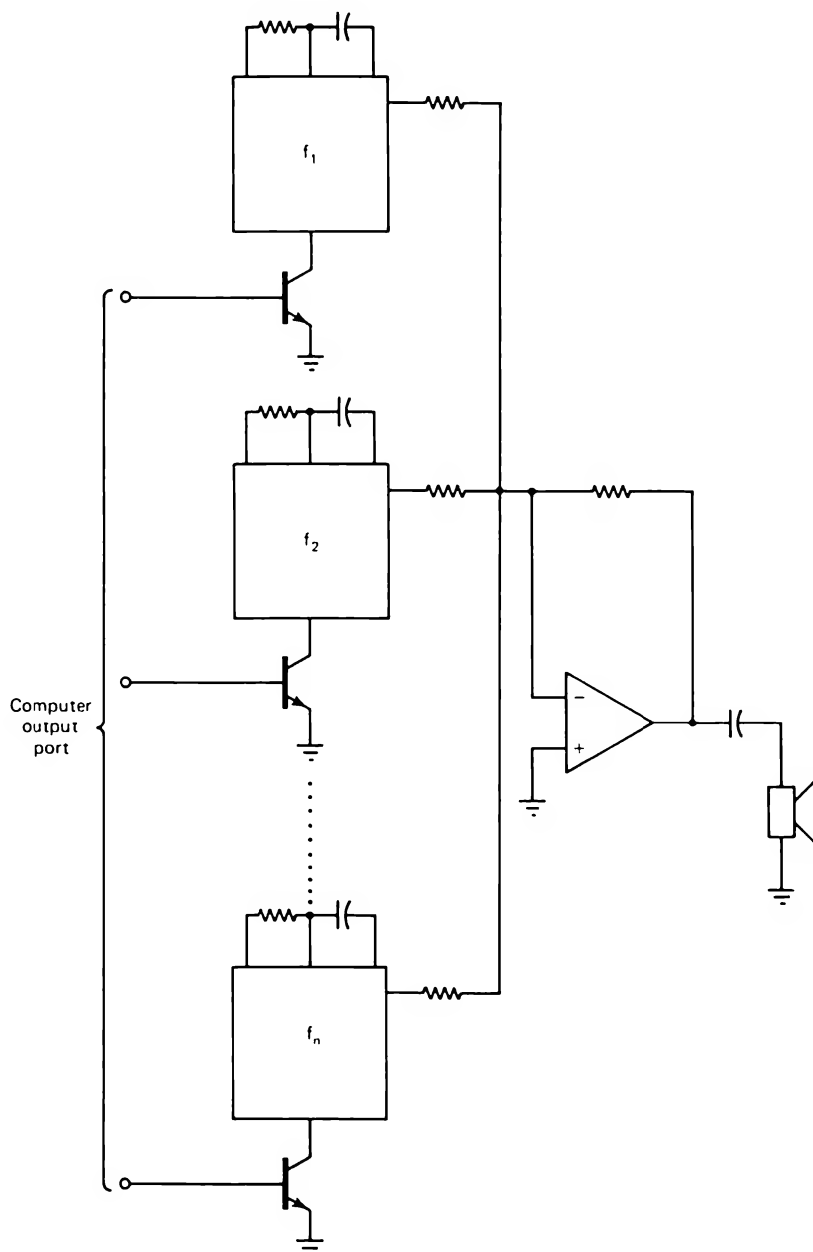


Figure 13-1 A computer output port can be used to turn on or off several pretuned oscillator circuits. The op-amp combines their output and drives the speaker.

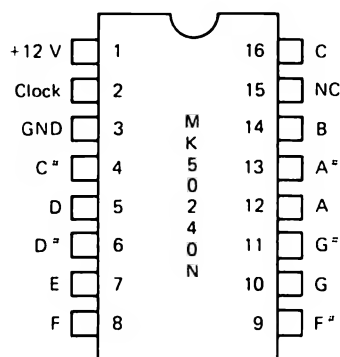
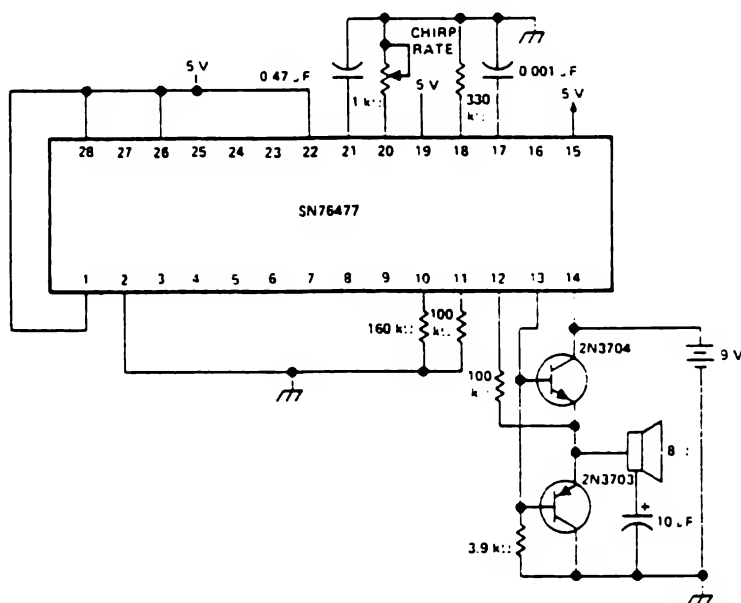


Figure 13-2 The Mostek MK50240N develops the 13 frequencies equivalent to the equal-tempered scale from a 2-MHz clock input.

Two types of complex sound generators are currently available. In the first, external capacitors and resistors establish basic circuit time constants and control the various waveshapes and tones. This type of circuit is generally hardwired to produce a particular sound or range of sounds. For example, the circuit in Fig. 13-3 produces the sound of a bird chirping, but by lowering the capacitance at pin 17, a “barking dog” sound is produced. Still other sounds can be produced by rewiring the circuit altogether. For exam-



For barking dog, the capacitor at pin 17 is changed to 15 pF to increase the frequency of the VCO.

Figure 13-3 Using the TI SN76477 to simulate a chirping bird. (Courtesy of Texas Instruments Incorporated.)

ple, the wiring needed to produce a gunshot or explosion effect is illustrated in Fig. 13-4.

The chip shown in both of these circuits is the Texas Instruments SN76477. This IC incorporates both digital and analog circuitry in one IC and includes a noise generator, voltage-controlled oscillator, super-low-frequency oscillator, noise filter, mixer, and envelope control circuit. Its particular function is programmed by wiring control pins high or low and connecting various resistor-capacitor combinations to the other pins. The SN76477 can be controlled by a microcomputer, but it is awkward and requires a substantial number of analog parts (resistors and capacitors).

The SN76488 is similar to the SN76477 and incorporates an on-chip audio amplifier capable of driving a small speaker. This eliminates the need for the two-transistor amplifier in Fig. 13-3 and 13-4.

A second type of complex sound generator is typified by the General Instrument AY-3-8910 shown in Fig. 13-5a and the TI SN76489A in Fig. 13-5b. These programmable sound generator ICs, like the Intel 8255, are meant to be controlled by a microcomputer. All their functions are con-

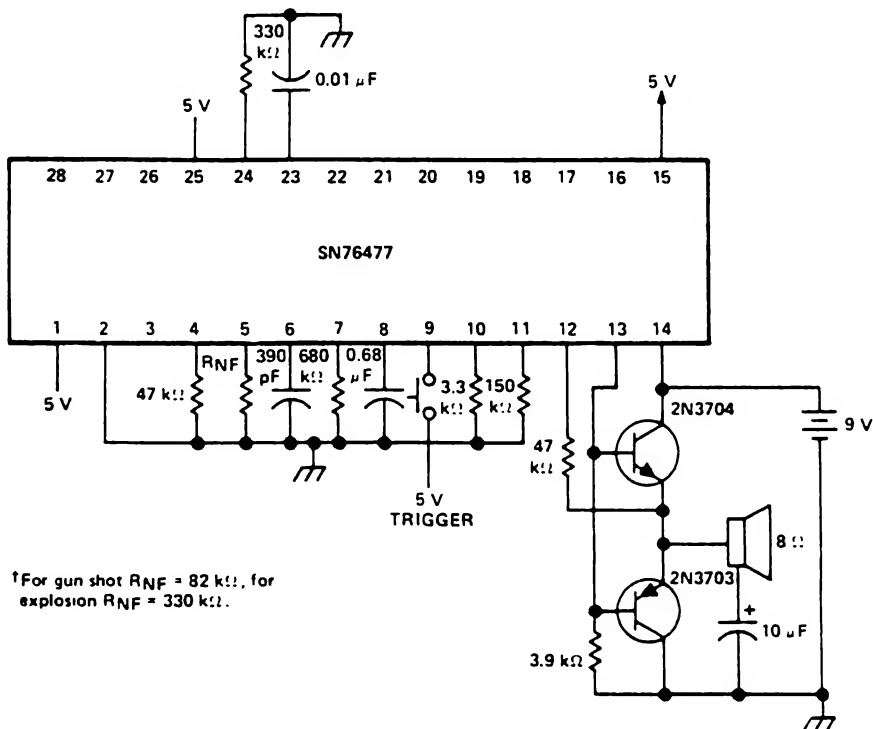
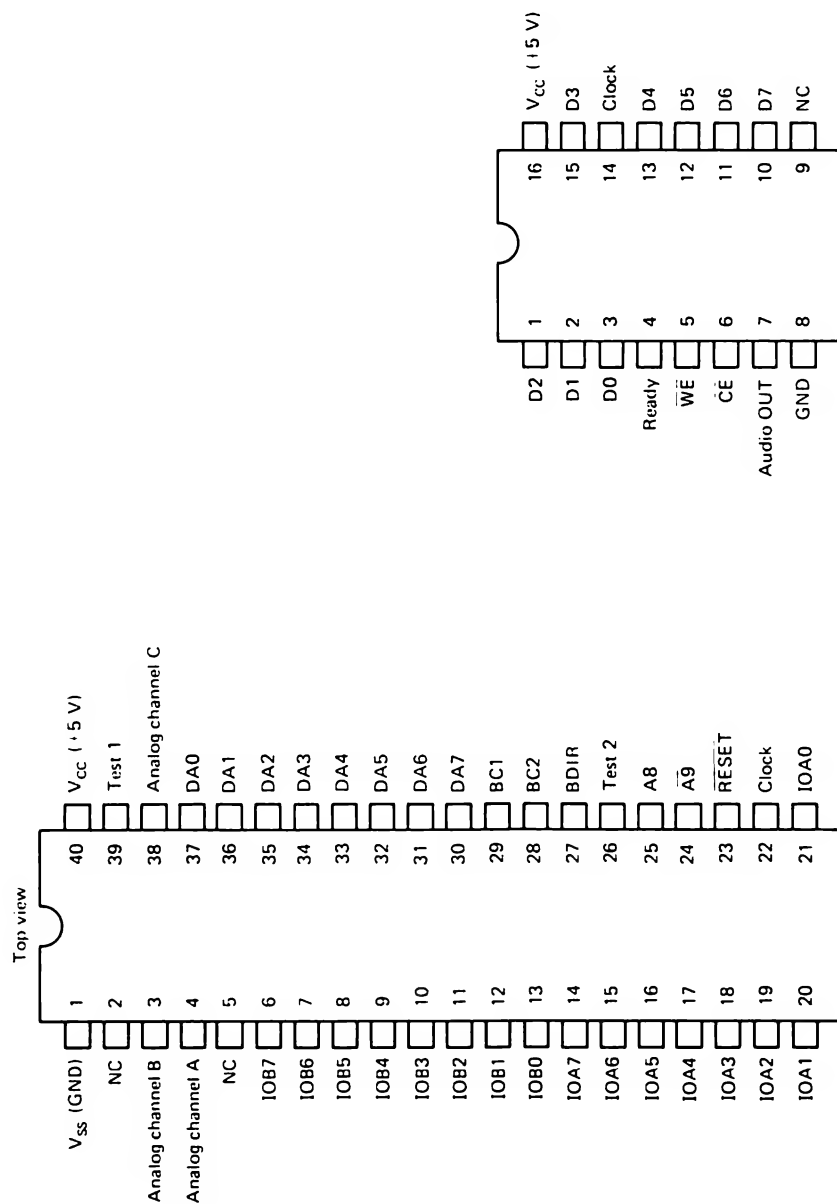


Figure 13-4 The circuit in Fig. 13-3 is rewired to produce a gunshot or explosion effect. Note the number of external resistors and capacitors required. (Courtesy of Texas Instruments Incorporated.)



(a) AY-3-8910

(b) SN76489A

Figure 13-5 Pin diagrams for the (a) AY-3-8910 and (b) SN76489A programmable complex sound generators.

trolled via an 8-bit bidirectional data bus connected to a microprocessor. No rewiring is necessary to change the sounds produced. This is because a new *control word* transmitted by the microprocessor accomplishes this function. In this experiment we will concentrate on the GI AY-3-8910 and its interface to the Apple.

The GI AY-3-8910

The General Instrument AY-3-8910 programmable sound generator (PSG) is an extremely powerful integrated circuit. Using a standard color TV crystal (3.58 MHz) for its time base, each of its three output channels can be programmed to produce a tone in the range 55 Hz to 224 kHz in 4069 discrete steps! The volume is also programmable and 16 different levels are possible. The three channels may be programmed to have different tones, allowing chords and resonance effects to be produced. The chip also contains a noise generator useful for simulating explosions, gunshots, and percussion instruments. In addition, the output envelope or waveshape can be controlled to allow variable attack and decay periods. For example, a ringing bell is simulated by an abrupt attack time but a longer decay period.

In addition to its sound capabilities, the AY-3-8910 features two programmable I/O ports that can be used similar to the I/O ports in the 8255. The AY-3-8912 is identical but features a single I/O port.

Interfacing the AY-3-8910 to the Apple

Note. The AY-3-8910 is a very complex IC. In explaining its operation, let us first treat it as a “black box” and show how it is interfaced to the Apple. After this is understood, we can study its internal register array and how to program these registers.

Figure 13-6 illustrates an interface circuit between the Apple and the PSG chip. There are several important aspects to this interface.

1. The *clock generator*. The PSG must receive a TTL-level clock signal at pin 22 with a frequency between 1 and 2 MHz. The circuit shown derives its time base from a 3.58-MHz TV crystal. The 4013 CMOS flip-flop provides a $\div 2$ function and the frequency applied to pin 22 is therefore 1.79 MHz.
2. The PSG *data bus*. All communication with the PSG is through an 8-bit bidirectional data bus at pins 30 through 37. Port A of the 8255 is programmed as an output port and data is output to the PSG with the command

POKE 49392,DATA

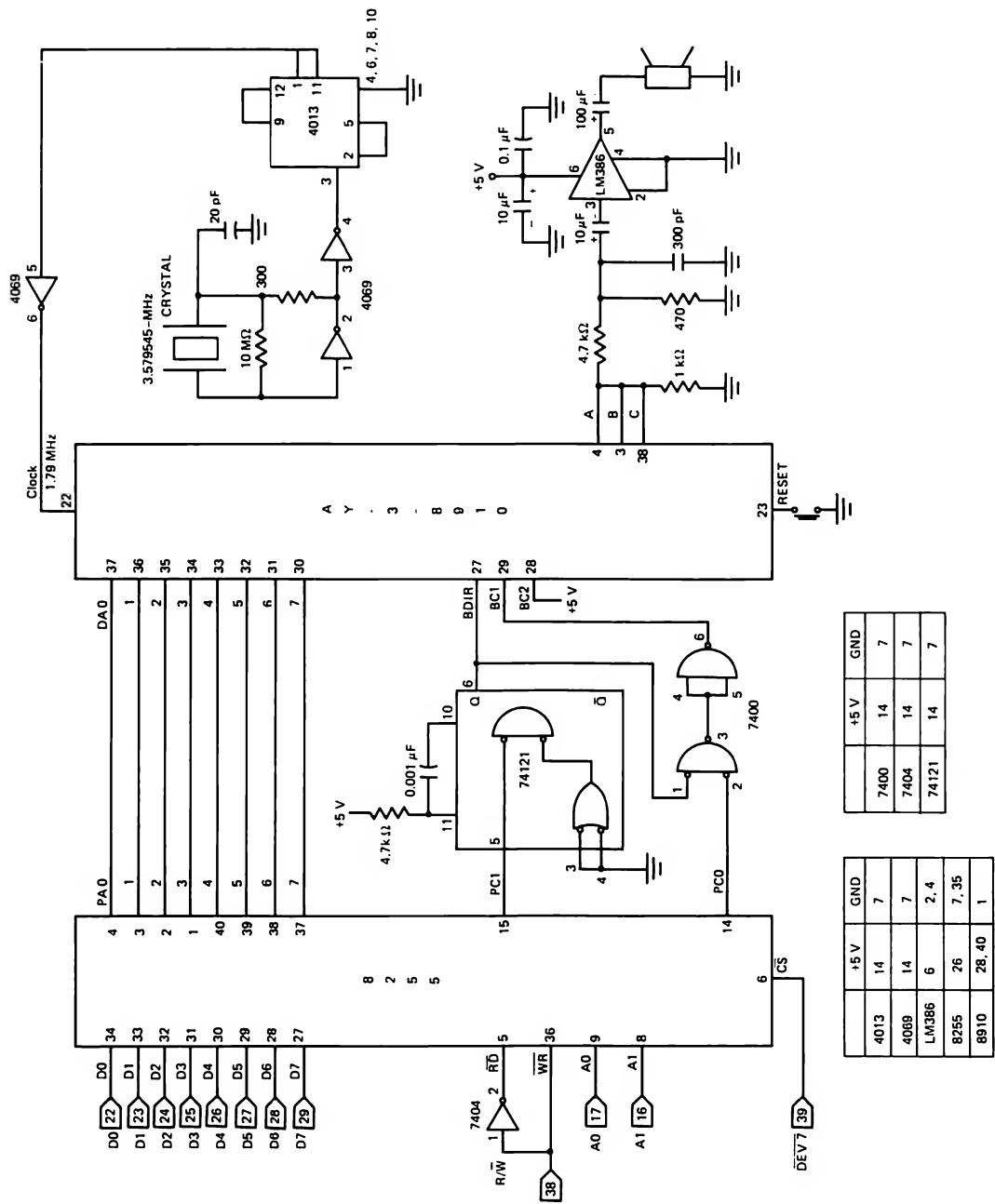


Figure 13-6 Interface circuit between the Apple and AY-3-8910 programmable sound generator. The 74121 one-shot is necessary to meet the *write data pulse width* specification.

Name	BDIR (pin 27)	BC1 (pin 29)	Function
Inactive	0	0	Hold. Chip should normally be held in this state.
Read	0	1	Read data from the PSG.
Write	1	0	Write data to the PSG.
Latch address	1	1	The data on the data bus should be latched and interpreted as an internal register address.

Figure 13-7 The four basic functions of the AY-3-8910 PSG as determined by the binary pattern on the control bus. Control pin BC2 (pin 28) is high for all combinations.

3. The *PSG control bus*. The control bus of the PSG determines the mode of operation of the chip and how it will interpret data on its data bus. Figure 13-7 lists the four possible modes. Normally, the chip is held in the *inactive* mode. When writing data, the address of the internal register desired must first be output and the control bus brought momentarily to the *latch address* state. Any subsequent data writes or reads will now be directed to or from this latched register. If it is desired to read or write from a different register, a new address must be output and the control bus momentarily sequenced through the latch address state.

Timing diagrams for the latch address, write data, and read data modes are illustrated in Fig. 13-8. When in the write mode care must be taken not to exceed the *write data pulse width* (t_{DW}), specified as 10 μ s by General Instrument. This is the purpose of the one-shot connected to PC1 in Fig. 13-6. When PC1 goes high, the one-shot output will pulse high for approximately 3 μ s (within the t_{DW} specification). Using BASIC it is not possible to pulse the PC1 line from low to high to low in less than 3 to 5 ms (3000 to 5000 μ s).

Example 13-1

What BASIC command must be given to cause the PSG control bus to sequence through each of the four modes listed in Fig. 13-7?

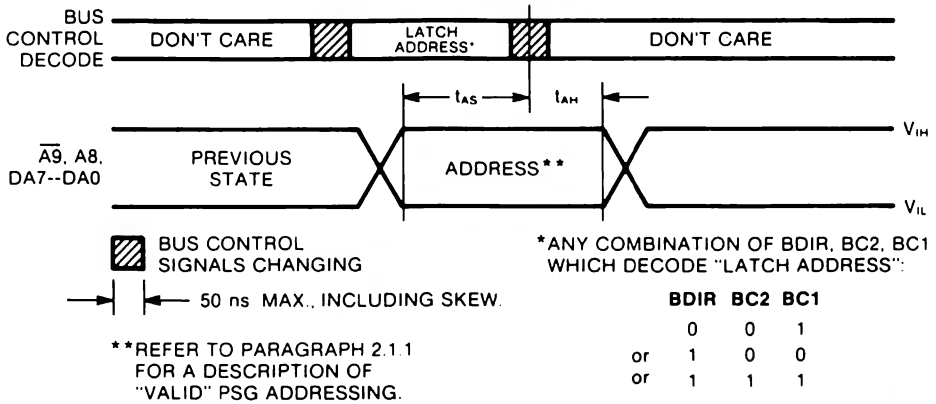
Solution Because the 8255 is memory mapped at ports 49392 through 49395, and the PSG control bus is connected to port C, all commands are of the form

POKE 49394,XX

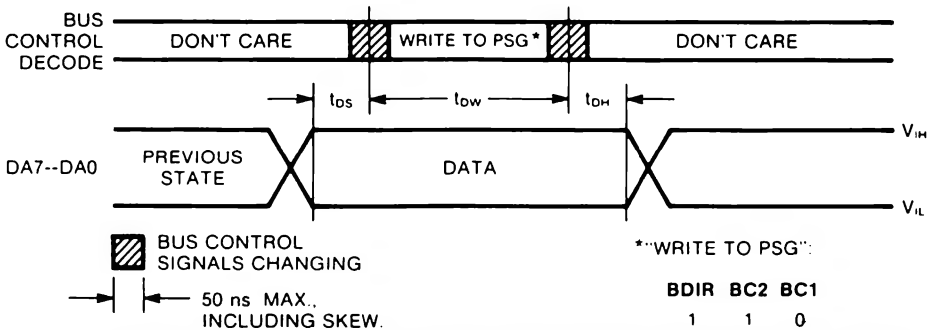
where port C corresponds to address 49394. The inactive state is entered with the command

POKE 49394,0 :REM INACTIVE

LATCH ADDRESS TIMING



WRITE DATA TIMING



READ DATA TIMING

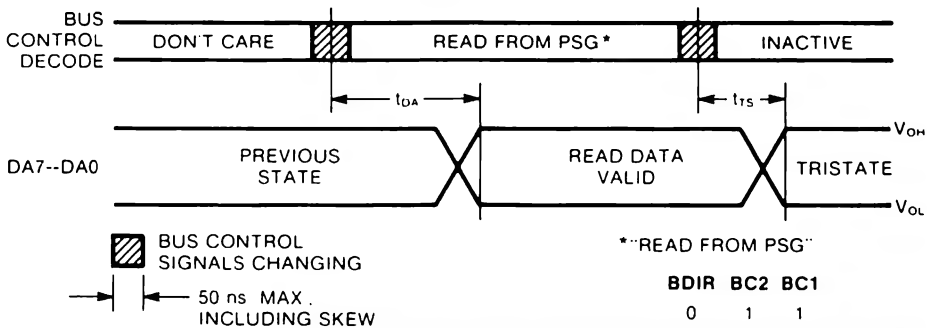


Figure 13-8 Timing relationship for the AY-3-8910 PSG. (Courtesy of General Instrument Corporation.)

PSG Software

Controlling the AY-3-8910 from BASIC requires that a specific sequence be followed to properly address and write into a desired register. Refer to the waveforms in Fig. 13-8 for the following sequence:

1. Output the register address.
2. Set the control lines to latch address (BDIR=1 BC1=1).
3. Set the control lines to inactive (BDIR=0 BC1=0).
4. Output the data for the selected register.
5. Set the control lines to write data (BDIR=1 BC1=0).
6. Set the control lines to inactive (BDIR=0 BC1=0).

Each time data is to be written to the PSG, the sequence outlined above must be followed. In addition, the control lines must not stay in the write data state for longer than 10 μ s (refer to the discussion on the control bus in the preceding section).

Example 13-2

Write a BASIC program that will input the desired register and data from the keyboard and output this information to the PSG using the proper sequence. Assume the hardware configuration shown in Fig. 13-6.

Solution

```

10 HOME
20 POKE 49395,128 :REM INIT PPI AND MAKE ALL PORTS OUTPUTS
30 INPUT "REGISTER AND DATA ";R,D
35 REM LATCH ADDRESS
40 POKE 49392,R: POKE 49394,3: POKE 49394,0
45 REM NOW WRITE DATA
50 POKE 49392,D: POKE 49394,2: POKE 49394,0
60 GOTO 30

```

Notice how the program follows the six steps listed previously. The PSG data bus is connected to port 49392 (8255 port A) and the control bus is controlled by bits PC0 and PC1 of port 49394 (8255 port C).

Now that the hardware is functional, all that remains is to determine what data to write to which register to make the PSG sound off! This is the hard part, but also the fun part!

Table 13-1 lists the functions of all 16 registers within the PSG. Seven distinct functions exist (actually only six functions control sound generation as registers 14 and 15 correspond to the two programmable I/O ports).

As Table 13-1 indicates, the PSG can be operated in two specific modes. In the first, called the *envelope mode*, register 13 controls the output ampli-

TABLE 13-1 THE 16 REGISTERS OF THE AY-3-8910 PSG^a

Register numbers	Function	Active in:	
		Envelope mode	Fixed-amplitude mode
0-5	Controls the tone on channels A, B, and C. Can be set between 55 Hz and 224 kHz.	Yes	Yes
6	Noise generator frequency. This frequency can be varied between 3.6 and 112 kHz.	Yes	Yes
7	Mixer control. This register allows any combination of channels to be enabled for noise and/or tones.	Yes	Yes
8-10	Amplitude control. The amplitude of each channel can be set to one of 16 levels.	No	Yes
11, 12	Envelope period control. These registers control the frequency or period of the output waveform in the envelope mode. They allow control of the attack and decay times.	Yes	No
13	Envelope shape. The actual envelope shape is set by the value in this register. Refer to Fig. 13-10 for specific waveshapes.	Yes	No
14, 15	I/O port data store	Yes	Yes

^aFrequencies specified assume a 1.79-MHz clock frequency.

tude and therefore *waveshape*. Figure 13-10 illustrates the various waveshapes possible. Notice that only 4 bits of this register are used (B0 through B3). Control of the waveshape allows for various sound effects. For example, writing a 0 to register 13 with the tone generator enabled and a long decay time will simulate a ringing bell.

The second mode of operation is called the *fixed-amplitude* mode. In this mode, the amplitude of each channel is set by registers 8 through 10 to one of 16 levels. No control of the waveshape is possible.

With either mode of operation, the chip can be enabled to produce a tone or noise signal or both on any of its three channels. Register 7 controls the mixing of these combinations.

Although I could try to describe the operation of each individual register, the best way to understand the capabilities of this IC is to assemble the test circuit in Fig. 13-6 and experiment with it yourself. Figure 13-11 summarizes the function of all 16 registers and is obtained from the *General Instrument Programmable Sound Generator Data Manual*. This manual is a

ENVELOPE SHAPE/CYCLE CONTROL

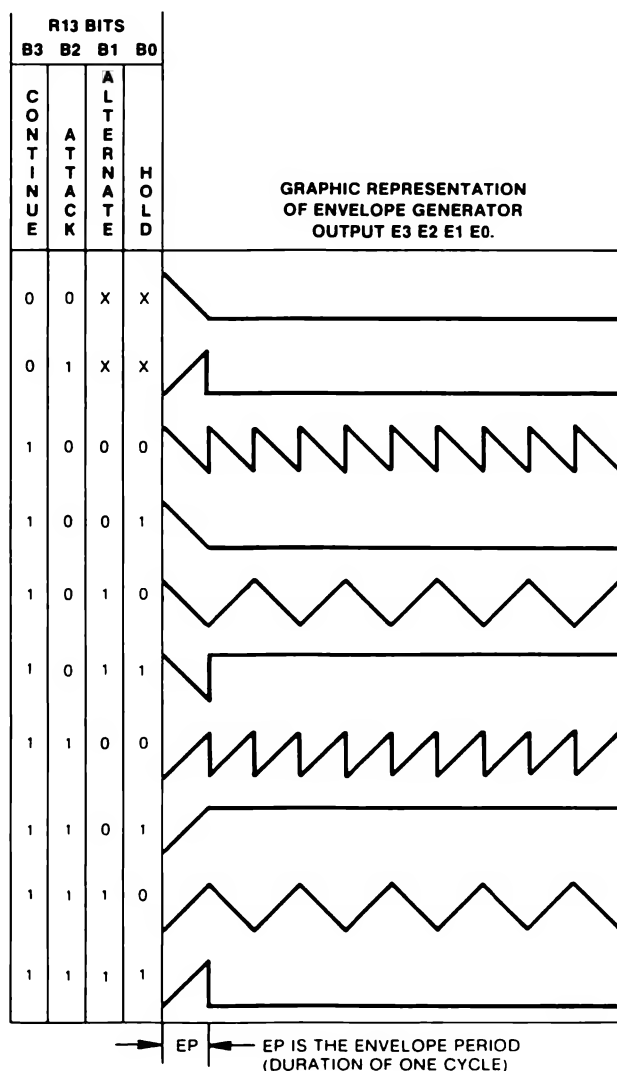


Figure 13-10 Ten separate waveshapes are possible when operating the AY-3-8910 PSG in the envelope control mode. Bits B0 through B3 of register 13 control this selection. (Courtesy of General Instrument Corporation.)

REGISTER		BIT							
		B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A Tone Period	8-BIT Fine Tune A							
R1						4-BIT Coarse Tune A			
R2	Channel B Tone Period	8-BIT Fine Tune B							
R3						4-BIT Coarse Tune B			
R4	Channel C Tone Period	8-BIT Fine Tune C							
R5						4-BIT Coarse Tune C			
R6	Noise Period				5-BIT Period Control				
R7	Enable	IN/OUT		Noise			Tone		
		IOB	IOA	C	B	A	C	B	A
R8	Channel A Amplitude				M★	L3	L2	L1	L0
R9	Channel B Amplitude				M★	L3	L2	L1	L0
R10	Channel C Amplitude				M★	L3	L2	L1	L0
R11	Envelope Period	8-BIT Fine Tune E							
R12		8-BIT Coarse Tune E							
R13	Envelope Shape/Cycle					CONT.	ATT.	ALT.	HOLD
R14	I/O Port A Data Store	8-BIT PARALLEL I/O on Port A							
R15	I/O Port B Data Store	8-BIT PARALLEL I/O Port B							

Tone period: $f_{\text{tone}} = \frac{f_{\text{clock}}}{16TP_{10}}$ $TP_{10} = 256CT + FT$
 CT = coarse tune register (1, 3, 5)
 FT = fine tune register (0, 2, 4)

Noise period: $f_N = \frac{f_{\text{clock}}}{16NP_{10}}$ $NP_{10} = \text{decimal equivalent of register 6}$

Envelope period: $f_E = \frac{f_{\text{clock}}}{256EP_{10}}$ $EP_{10} = 256CT + FT$
 CT = coarse tune register (12)
 FT = fine tune register (11)

* Mode bit: M = 1 for fixed amplitude mode
 M = 0 for envelope generator mode

Figure 13-11 Summary of all 16 AY-3-8910 PSG register functions.
 (Courtesy of General Instrument Corporation.)

necessity for working with this chip and can be obtained by writing to General Instrument Corp., Microelectronics Division, 600 W. John Street, Hicksville, NY 11802.

PROCEDURE

Step 1. Refer to Fig. 13-6 and carefully wire this circuit on your breadboard. This is a complex circuit requiring eight ICs, so take your time to be sure each connection is correct. If you have an audio amplifier, the LM386 portion of the interface can be skipped and the output from pins 3, 4, and 38 across the 1-k Ω resistor connected to your amplifier. Take care when handling the AY-3-8910, as it is a static-sensitive MOS device.

Step 2. Test your hardware by running the program given in Example 13-2. Type the following responses to the "REGISTER AND DATA" prompt:

1,1	Set coarse tone for channel A to 1.
7,62	Enable tone on channel A only.
8,10	Set the volume for channel A to 10.

After typing 8,10 you should hear a low-pitched tone. Try changing the volume by writing new data to register 8. Change the tone by writing to register 1.

Question 13-1. What response is needed to set the tone to its lowest value and the volume to maximum?

Question 13-2. Type

1,0
0,100

Explain the result.

Step 3. Type

2,101	Write 101 to channel B fine tune tone register.
7,60	Enable both channels A and B.
8,10	Set the volume for channel A to 10.
9,10	Set the volume for channel B to 10.

Question 13-3. Explain the effect produced in step 3.

Step 4. Reset the PSG by touching pin 23 to ground. With the test program still running type:

7,7	Enable noise on all three channels.
8,10	Set the volume of all three channels to 10.
9,10	
10,10	

You should hear a rushing sound. Try writing numbers to register 6 between 0 and 31. This will change the noise generator frequency.

Step 5. With the pattern of step 4 established, try the following:

6,15	Set the noise frequency to midvalue.
8,16	Enable the envelope generator mode for all three channels.
9,16	
10,16	
12,16	Set a short decay time.
13,0	Set the envelope decay for one cycle only.

After typing 13,0 you should hear a single “*gunshot!*”

Step 6. Change the registers by typing:

6,0	Set the noise frequency to its lowest value.
12,56	Set a longer decay time.
13,0	Set the envelope decay for one cycle only.

You should hear the sound of an “*explosion.*”

Question 13-4. With a small change the PSG will now produce the sound of a ringing bell. What changes are needed?

Step 7. Modify the answer to Question 13-4 by typing 13,8 in place of 13,0. The bell should now ring on its own (refer to Fig. 13-10 to see why). Changing the data written to register 12 will change the delay time.

Step 8. Write a program to simulate a *whistling bomb* sound effect using the explosion produced in step 6. A solution is provided at the end of this experiment.

Step 9. The following program converts the Apple into an *electronic organ*. Keys A through K correspond to one consecutive octave of notes with C# at W, D# at E, F# at T, G# at Y, and A# at U. The computer will save your notes (and mistakes!) and display the current number of notes in memory on the screen. The offset allows a vibrato effect.

```

10 HOME: DIM D(200),N$(500)
20 PRINT " APPLE AY-3-8910 ELECTRONIC ORGAN"
30 R=7: D=62: GOSUB 1000 :REM ENABLE TONE FOR A
40 R=8: D=10: GOSUB 1000 :REM VOLUME ON A TO 10
50 FOR K=1 TO 25 :REM THIS IS A SHORT TUNE
60 R=0
70 FOR D=48 TO 54: GOSUB 1000: NEXT D
80 FOR D=54 TO 48: GOSUB 1000: NEXT D
90 NEXT K
100 READ N1,N2 :REM GET ASCII KEY &
TONE CODE

```

```

110 IF N1=999 THEN 140
120 D(N1)=N2                                :REM ASSIGN TONE CODE TO
                                           ARRAY
130 GOTO 100
140 POKE 49395,128                          :REM INIT PPI
150 R=7: D=63: GOSUB 1000                  :REM DISABLE ALL CHANNELS
160 VTAB(3)
170 INPUT "ENTER OFFSET OR 0: ";BOFF
180 VTAB(5)
190 INPUT "FIXED OR ENVELOPE MODE: ";V$
200 PRINT
210 IF LEFT$(V$,1)="F" THEN 280
220 D=16: R=8: GOSUB 1000                  :REM ENABLE ENVELOPE MODE
230 R=9: GOSUB 1000
240 VTAB(7)
250 INPUT "ENTER DECAY VALUE (0-255): ";DECAY
260 R=12: D=DECAY: GOSUB 1000
270 GOTO 330
280 VTAB(7): INPUT "WHAT VOLUME LEVEL (0-15): ";V
290 R=8: D=V: GOSUB 1000
300 R=9: GOSUB 1000
310 R=0: D=0: GOSUB 1000                  :REM BE SURE TONE IS OFF
320 R=2: GOSUB 1000
330 VTAB(9)
340 PRINT "I WILL SAVE YOUR NOTES."
350 PRINT: PRINT "TYPE Q TO QUIT."
360 R=7: D=60: GOSUB 1000                  :REM ENABLE TONE ON A AND B
370 GET N$
380 IF N$="Q" THEN 3000
390 N$(I)=N$: I=I+1: VTAB(22): HTAB(36): PRINT I;
400 N=ASC(N$)
410 R=0: D=D(N): GOSUB 1000                :REM PLAY TONE ON A
420 R=2: D=D(N)-BOFF: GOSUB 1000          :REM PLAY OFFSET ON B
430 R=13: D=0: GOSUB 1000                 :REM DECAY ONE CYCLE ONLY
440 GOTO 370
1000 IF D>255 OR D<0 THEN D=0
1010 POKE 49392,R: POKE 49394,3: POKE 49394,0
1020 POKE 49392,D: POKE 49394,2: POKE 49394,0
1030 RETURN
3000 PRINT
3010 INPUT "ENTER THE TEMPO (0-500): ";T
3020 FOR J=0 TO I-1
3030 N=ASC(N$(J))
3040 R=0: D=D(N): GOSUB 1000              :REM PLAY NOTE ON A
3050 R=2: D=D(N)-BOFF: GOSUB 1000        :REM PLAY OFFSET NOTE
3060 R=13: D=0: GOSUB 1000               :REM DECAY ONE CYCLE ONLY
3070 FOR K=1 TO T: NEXT K                :REM TEMPO
3080 NEXT J

```

```

3090 FOR K=1 TO 1000: NEXT K           :REM ALLOW LAST NOTE TO DIE
3100 VTAB(4): GOTO 150
4000 DATA 65,214,87,202,83,190,69,180,68,170,70,160,84,151
4010 DATA 71,143,89,135,72,127,85,120,74,113,75,107,999,999

```

Step 10. Reread the objectives listed at the beginning of this experiment. If these points are not clear to you, restudy the "Discussion" and "Procedure" sections.

SOLUTIONS TO QUESTIONS

13-1

1,15 Lowest tone frequency.
 8,15 Highest volume.

13-2. The coarse tone register for channel A is set to 0 and the fine tune register is set to 100. The new tone produced is of a much higher frequency.

13-3. Because the channel A tone register (register 0) = 100, but the channel B tone = 101 (register 2), a resonance effect is heard.

13-4

0,100 Set all three channels for a high-pitched tone.
 2,100
 4,100
 7,56 Enable tone only for channels A to C.
 13,0 One decay cycle only.

Step 8 (solution)

```

10 HOME
20 POKE 49395,128           :REM INIT PPI
30 R=7:D=62: GOSUB 500      :REM ENABLE TONE ON A ONLY
40 R=8:D=15: GOSUB 500      :REM MAXIMUM VOLUME
50 FOR J=48 TO 192          :REM SWEEP EFFECT
60 R=0:D=J: GOSUB 500
70 NEXT J
80 R=6:D=15: GOSUB 500      :REM SET NOISE TO MIDVALUE
90 R=7:D=7: GOSUB 500       :REM ENABLE NOISE ON ALL CHANNELS
100 R=8:D=16: GOSUB 500     :REM ENVELOPE MODE
110 R=9: GOSUB 500
120 R=10: GOSUB 500
130 R=12:D=56: GOSUB 500    :REM TIME DELAY FOR EXPLOSION
140 R=13: D=0: GOSUB 500    :REM DECAY ONE CYCLE ONLY
150 INPUT A$                :REM WAIT TO DO IT AGAIN
160 GOTO 30
500 POKE 49392,R: POKE 49394,3: POKE 49394,0
510 POKE 49392,D: POKE 49394,2: POKE 49394,0
520 RETURN

```

APPENDIX A

WIRING THE VECTOR CARD

Interfacing to the Apple computer is accomplished via the eight *peripheral connectors* located across the rear of the main circuit board (motherboard). These eight connectors are also referred to as slots 0 through 7.

Access to these connectors is obtained by removing the computer top cover. Any interconnecting cables are routed through special openings in the rear of the case.

SIGNAL DESCRIPTIONS

The signal descriptions of each of the 50 pins of a typical peripheral connector are given in Fig. A-1. Interfacing to these 50 pins, we can design digital circuits whose function will be controlled by programs running on the Apple.

For the experiments in this book, access to these pins is obtained by plugging a specially wired *Vector 4609 plugboard* into one of the slot connectors. The outline of such a card is shown in Fig. A-2 and two photos are provided in Fig. I-2a and b. The Vector plugboard mates with the slot connector and also provides a mate for a 40-pin *card edge connector* at its rear. This can be seen in Fig. A-2. As described in Part 1, a 40-conductor ribbon cable with card edge connector on one end and socket connector on the other is used to bring these signals to the breadboard.

Although the plugboard can provide only 40 connections, this turns out to be more than adequate for the purposes of this book. Jumper wires are used to wire the slot connector end of the card to the 40-pin card edge connector end.

Figure A-3 lists the connections that must be made between the two connector receptacles and summarizes the resulting pinning at the socket connector end of the ribbon cable.

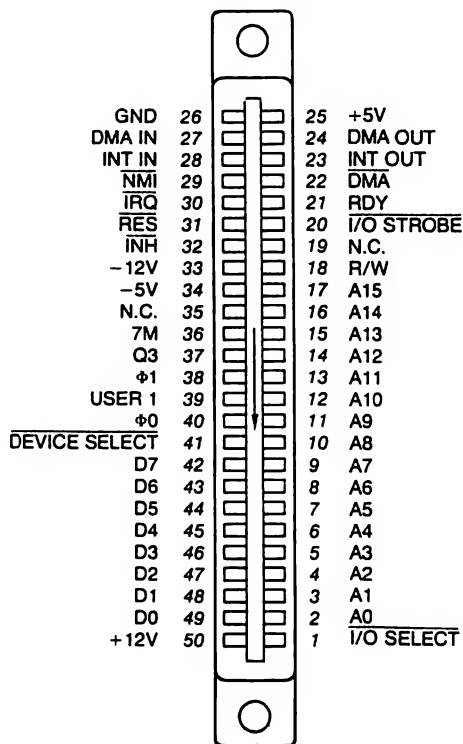


Figure A-1 Pin connections for a typical peripheral connector on the Apple motherboard. (Reprinted from the Apple II Reference Manual with the permission of Apple Computer, Inc.)

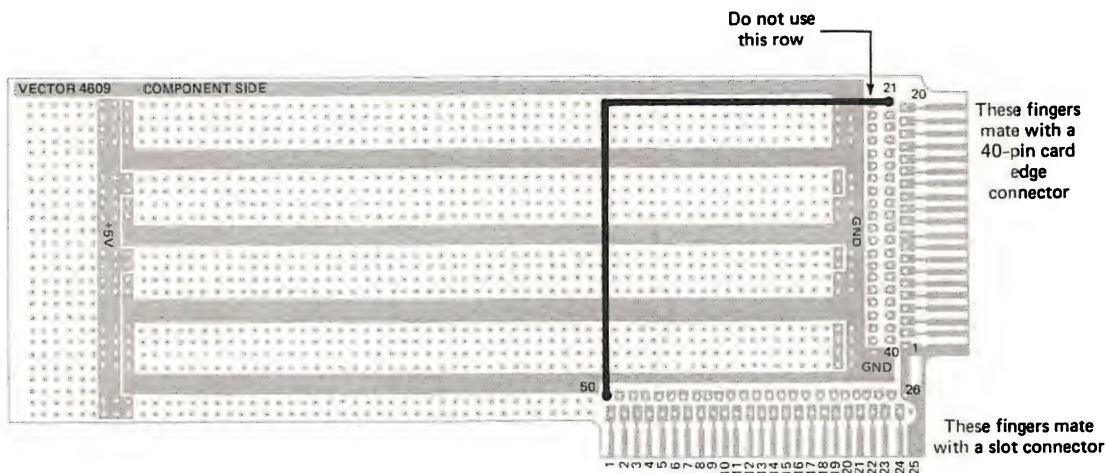


Figure A-2 Outline of the Vector 4609 plugboard. The component side is shown with one of the typical jumper wire connections highlighted (pin 50 to pin 21).

Description	Apple pin number	Vector pin number
Data bus D0	49	22
Data bus D1	48	23
Data bus D2	47	24
Data bus D3	46	25
Data bus D4	45	26
Data bus D5	44	27
Data bus D6	43	28
Data bus D7	42	29
+5 V	25	1
+12 V	50	21
-12 V	33	30
-5 V	34	31
Ground	26	40
Address bus A0	2	17
Address bus A1	3	16
Address bus A2	4	15
Address bus A3	5	14
Address bus A4	6	13
Address bus A5	7	12
Address bus A6	8	11
Address bus A7	9	10
Address bus A8	10	9
Address bus A9	11	8
Address bus A10	12	7
Address bus A11	13	6
Address bus A12	14	5
Address bus A13	15	4
Address bus A14	16	3
Address bus A15	17	2
Device select	41	39
R/ \overline{W}	18	38
$\Phi 0$	40	37

(a)

20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
—	—	—	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	+5 V
<hr/>																			
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
+12 V	D0	D1	D2	D3	D4	D5	D6	D7	-12 V	-5 V	—	—	—	—	—	$\Phi 0$	R/ \overline{W}	\overline{DS}	GND

(b)

Figure A-3 (a) Pin number correspondence between the Appie and the Vector plugboard. (b) Summary chart handy for locating pin numbers when doing the experiments.

WIRING THE CARD

Before wiring the Vector card, please note the following.

1. Orient the card with the component side up.
2. Use solid No. 22 or 24 gauge *insulated* wire.
3. Use Fig. A-3 to determine the source and destination of each wire.
4. *Do not solder* the connections until all wires have been mounted on the board. Mount these wires as neatly as possible.
5. The +5 V and ground connections require no jumper wires.
6. Once all connections are made (but not soldered), connect the ribbon cable to the card. Now using an ohmmeter, check the socket connector end of the cable for proper continuity to each pin of the peripheral connector. If you do not have an ohmmeter, carefully check each connection to be sure that it is correct.
7. Solder all connections.

Once wired, and with the ribbon cable attached, mount the card in the Apple. Refer to Fig. I-2a and notice that the cable should be connected so as to exit to the *left* of the Vector card when viewing the computer from the front. Route the excess cable length through one of the openings in the rear of the cabinet.

The socket connector end of the cable should then be secured to your breadboard. Steps 1 through 3 of Experiment 1 detail the final steps necessary to make your breadboard “operational.”

APPENDIX B

PARTS LIST

Tables B-1 through B-4 list all the parts required to perform the 13 experiments in this book. The following information is supplied:

1. Part number
2. Description of the part
3. Quantity needed to do any one experiment
4. Supplier code
 - RS Radio Shack
 - J Jameco Electronics, 1355 Shoreway Road, Belmont, CA 94002
 - D Digi-Key Corp., P.O. Box 677, Thief River Falls, MN 56701
 - P Priority One Electronics, 9161 Deering Avenue, Chatsworth, CA 91311

Most of the parts are noncritical and slight variations should present no problems in the experiments (for example, substituting a 330- Ω resistor for a 300- Ω resistor). Also, the 74LS series of TTL gates may be substituted for any of the standard 7400 part numbers given. If you are not sure about a particular parts description, refer to the text for that experiment, as it often contains more detail about the components required. A good idea would be to write to the companies listed and request a parts catalog. Many other suppliers also exist and you should consult any of the popular hobbyist magazines.

TABLE B-1 PARTS LIST: INTEGRATED CIRCUITS

Part number	Description	Quantity	Supplier	Number per experiment										
				1	2	3	4	5	6	7	8	9	10	11 12 13
7400	Quad NAND gates	1	RS J D P	1					1					1
7402	Quad NOR gates	1	RS J D P							1				
7404	Hex inverters	2	RS J D P	1	1	2	1	1	1	1	1	1	1	1
7420	Dual NAND gates	1	J D P	1										
7427	Triple NOR gates	1	J D P		1									
7432	Quad OR gates	1	RS J D P	1	1									
7476	Dual JK flip-flop	1	J D P	1	1	1	1							
7485	4-bit comparator	1	J D P											
74100	8-bit latch	1	J										1	
74121	One-shot	1	J D P											1
74LS244	Tri-state buffers	1	RS J D P				1							
4013	Dual flip-flop (CMOS)	1	RS J D P											1
4069	Hex inverters (CMOS)	1	RS J D P											1
8255	PPI	1	J D P					1	1	1	1		1	1
6820	PIA	1	J P						1					
6850	ACIA	1	J P											1
MC1408	8-bit DAC	1	J P							1				
AY-3-8910	PSG	1	see Exp. 13											1
TIL111 or 4N25	Opto-coupler	1	RS J						1					
ADC0809	8-bit ADC	1	J D									1		
LM334	Temperature sensor	1	RS J D						1		1			
LM339	Quad comparators	1	RS J D P						1					
LM386	Audio amplifier	1	RS J D							1				1
LM555	Timer	1	RS J D P											1
LM567	Tone decoder	1	RS J D							1				
LM1458	Dual op-amp	1	RS J D P								1	1		

TABLE B-2 PARTS LIST: RESISTORS

Description	Quantity	Supplier	Number per experiment												
			1	2	3	4	5	6	7	8	9	10	11	12	13
68- Ω resistor	1	RS J D P											1		
100- Ω resistor	1	RS J D P								1					
150- Ω resistor	1	RS J D P								1					
180- Ω resistor	8	RS J D P	1	1	8		6	6			1		2	1	
220- Ω resistor	1	RS J D P							1	1		1			
270- Ω resistor	1	RS J D P									1				
300- Ω resistor	1	J D P													1
330- Ω resistor	1	RS J D P								1					
470- Ω resistor	1	RS J D P													1
1-k Ω resistor	10	RS J D P				8	2	2	3	2	2	3	10	1	1
4.7-k Ω resistor	2	RS J D P													2
10-k Ω resistor	2	RS J D P							2			1			
330-k Ω resistor	1	J D P										1			
10-M Ω resistor	1	RS J P													1
10-k Ω potentiometer	2	RS J D P							2	2	1	1			
50-k Ω potentiometer	1	RS J D P										1		1	
Photo cell	1	RS							1						

TABLE B-3 PARTS LIST: CAPACITORS

Description	Quantity	Supplier	Number per experiment												
			1	2	3	4	5	6	7	8	9	10	11	12	13
20-pF capacitor	1	J D													1
100-pF capacitor	1	RS J D									1				
300-pF capacitor	1	J D													1
680-pF capacitor	1	J D										1			
0.001- μ F capacitor	1	RS J D													1
0.0047- μ F capacitor	1	RS J D								1					
0.005- μ F capacitor	1	RS J D								1					
0.01- μ F capacitor	1	RS J D												1	
0.033- μ F capacitor	1	J D								1					
0.05- μ F capacitor	1	RS J D								1					
0.1- μ F capacitor	1	RS J D								1					1
10- μ F capacitor	3	RS J D	1	1	1	1	1	1	1	2	1	3	1	1	3
22- μ F capacitor	1	RS J D												1	
100- μ F capacitor	1	RS J D								1					1

TABLE B-4 PARTS LIST: MISCELLANEOUS

Part	Description	Quantity	Supplier	Number per experiment												
				1	2	3	4	5	6	7	8	9	10	11	12	13
LED	Red	8	RS J D P	1	1	8		2	2		1	1		1	1	
LED	Green	2	RS J D P					2	2					1		
LED	Yellow	2	RS J D P					2	2							
Switch	PBNO	2	RS J D					2	2					2		
Switch	Magnetic	1	RS							1	1					
Switch	8-position DIP	1	RS J P		1	1								1		
Buzzer	Piezoelectric	1	RS J D								1					
Transistor	<i>npn</i>	1	RS J D P								1					
Transducer	40 kHz	2	see Exp. 8								2					
Relay	SPDT DIP	1	RS J D								1					
Crystal	3.579545 MHz	1	RS J D P													1
Diode	General purpose	1	RS J D P								1					
Diode	6.2-V zener (optional)	1	RS J D P										1			
Display	Seven-segment (common anode)	1	RS J D P									1				

APPENDIX C

BINARY AND DECIMAL NUMBERS

All digital computers work with electrical circuits that can be considered *ON* or *OFF*. Usually, this is referred to as a *1* or a *0*, or a *high* and a *low*. Because of this *ON-OFF* nature, the *binary* number system is used to represent data and addresses within the computer.

Just as decimal is a base *ten* number system, binary is a base *two* number system. Each binary digit or *bit* has a value or weight that is a *factor of 2* higher than the digit to its right. The value of any digit is found as the base (2) raised to the power corresponding to its position from the right. The first position is considered zero. For example,

$$\begin{aligned} 1101 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 8 + 4 + 0 + 1 = 13_{10} \end{aligned}$$

Binary numbers are converted to their decimal equivalents by simply adding the appropriate binary weights. Table C-1 lists the binary value for each of the first 16-bit positions. Notice that when moving from right to left, each bit is a factor of 2 higher than its rightmost neighbor.

The Apple uses 8-bit words and some typical examples are:

$$00001011 = 8 + 2 + 1 = 11_{10}$$

$$00100110 = 32 + 4 + 2 = 38_{10}$$

$$11000111 = 128 + 64 + 4 + 2 + 1 = 199_{10}$$

$$11111111 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255_{10}$$

TABLE C-1 POWERS OF 2 FOR THE FIRST 16 BIT POSITIONS

Bit position	Binary weight
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16,384
15	32,768

When converting from decimal to binary, the highest remaining power of 2 is repeatedly *subtracted* from the decimal number. For example,

$$\begin{array}{rcl}
 (2^4) & - \frac{25_{10}}{16} & \text{(highest power of 2 in 25)} \\
 & \underline{9} & \\
 (2^3) & - \frac{8}{1} & \text{(highest power of 2 in 9)} \\
 & \underline{1} & \\
 (2^0) & - \frac{1}{0} & \text{(highest power of 2 in 1)}
 \end{array}$$

The answer is then the binary number corresponding to $2^4(16) + 2^3(8) + 2^0(1)$ or 11001. Other examples are:

$$58 = 2^5(32) + 2^4(16) + 2^3(8) + 2^1(2) = 111010$$

$$147 = 2^7(128) + 2^4(16) + 2^1(2) + 2^0(1) = 10010011$$

$$250 = 2^7(128) + 2^6(64) + 2^5(32) + 2^4(16) + 2^3(8) + 2^1(2) = 11111010$$

APPENDIX D

BASIC LOGIC GATES

There are only *three* basic logic functions used in any digital system. These are the *AND*, *OR*, and *NOT* functions. Schematically these are represented as logic gates and illustrated in Fig. D-1.

The *AND* gate requires A AND B to be high in order for the output C to be high. The *OR* gate requires A OR B to be high in order for its output C to be high. The *NOT* gate or inverter simply inverts the value of its input: a 1 in yields a 0 out, and a 0 in yields a 1 out.

Although the *AND* and *OR* gates in Fig. D-1 have only two inputs (A and B), logic gates with more than two inputs are also available.

Two common variations of the *AND* and *OR* gate are the *NAND* (*NOT-AND*) and *NOR* (*NOT-OR*) gates. These are illustrated in Fig. D-2. From their truth tables (and logic symbols) it can readily be seen that these gates simply provide the *inverted* output of their *AND* and *OR* gate equivalents. Thus a *NAND* gate has a 0 output when inputs A AND B are high, while the *NOR* gate provides a 0 output when inputs A OR B are high.

Although the truth tables for the various logic gates can (and probably should) be memorized, their schematic symbols also suggest a word interpretation. Refer to Fig. D-3a for the *AND* gate. If we use the convention that a circle represents an inversion or low (0) logic level, then the standard *AND* gate symbol says in words: When inputs A and B are high, output C will be high. However, by referring to the *AND* gate truth table in Fig. D-1a, we can also see that any time A OR B is a 0, the output is also a 0. This suggests the second symbol drawn in Fig. D-3a. The word interpretation is: When A OR B is low, the output C is low. Note how the small circles are used and interpreted for this alternate *AND* gate symbol. Figures D-3b-(d) illustrate the corresponding interpretation for the *OR*, *NAND*, and *NOR* gates and their alternate symbols.

You might wonder why these other logic symbols are used. The answer

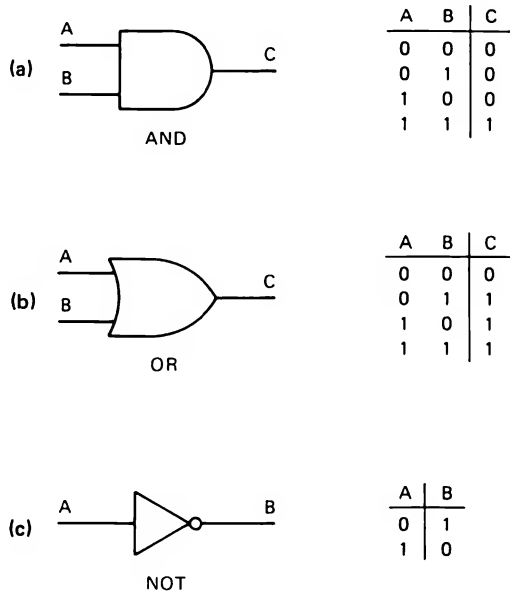


Figure D-1 The three basic logic gates and their truth tables: (a) AND; (b) OR; (c) NOT.

is that often logic signals are *active low* (go to a 0 level when they are present) and we may need to perform any of the various logic functions on these active low signals. Figure D-4 illustrates this case. In this figure we need to determine when two active low signals (note the \sqcap symbol), $\overline{\text{SEL219}}$ AND $\overline{\text{IN}}$, are *both* low and then produce an active high output signal (IN 219).

Although the circuits in parts a and b of Fig. D-4 are equivalent and both use the same two-input NOR gate, Fig. D-4a is much easier to understand in terms of what the circuit is doing than is the equivalent circuit in Fig. D-4b.

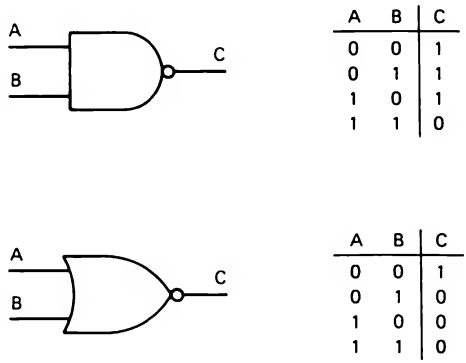


Figure D-2 Truth tables and logic symbols for the (a) NAND and (b) NOR gates.

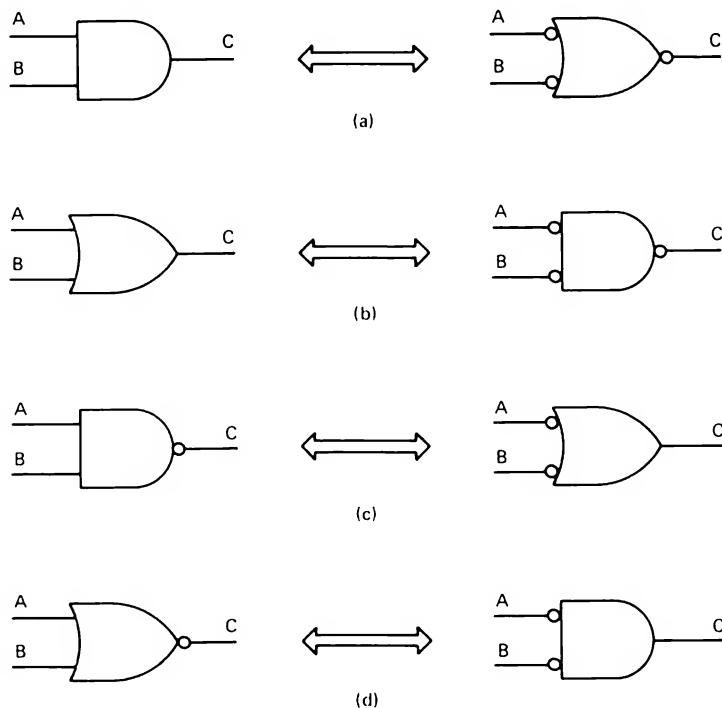


Figure D-3 The primary and alternate (active low) logic symbols are shown for (a) the AND gate, (b) the OR gate, (c) the NAND gate, and (d) the NOR gate.

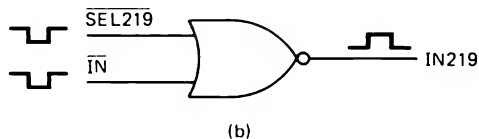
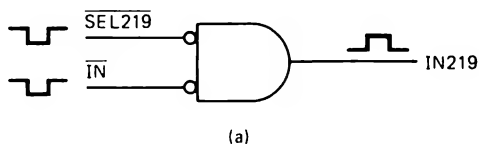


Figure D-4 A NOR gate is used to detect when $\overline{\text{SEL219}}$ AND $\overline{\text{IN}}$ are both low and generate an active high output pulse. Both circuits (a) and (b) are equivalent, but only (a) gives the proper word interpretation.

APPENDIX E

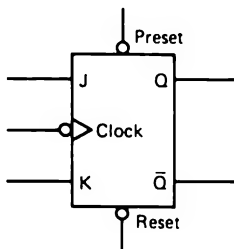
JK FLIP-FLOP

A flip-flop is a logic element that is used to *store* or remember a logic condition. Typically, a *clock* signal is applied to the flip-flop and at the instant this clock occurs, the flip-flop *latches* the binary signal it is monitoring.

There are several different types of flip-flops, but the most versatile is the *JK* type. The symbol for this flip-flop is illustrated in Fig. E-1. There are two sets of inputs, referred to as the *asynchronous* (PRESET and RESET) and *synchronous* (J and K) inputs.

The flip-flop responds to the asynchronous inputs *immediately*, independent of the clock signal. Studying the first three entries in the truth table, when the PRESET input is low, the Q output is *set* or high. Similarly, when the RESET input is low, the Q output is *reset* or low. Because it is not logical to try to set and reset the flip-flop at the same time, the PRESET and RESET inputs should not both be low at the same instant.

The synchronous inputs, J and K, are used only when the PRESET and RESET inputs are both high. This corresponds to the last four entries in the table. Using J and K, the flip-flop output will change only when the clock



Asynchronous inputs		Synchronous inputs		Output
Preset	Reset	J	K	Q_{t+1}
0	1	X	X	1
1	0	X	X	0
0	0	X	X	Not allowed
1	1	0	0	No change
1	1	0	1	0
1	1	1	0	1
1	1	1	1	Toggle

Figure E-1 Truth table and logic symbol for the JK flip-flop.

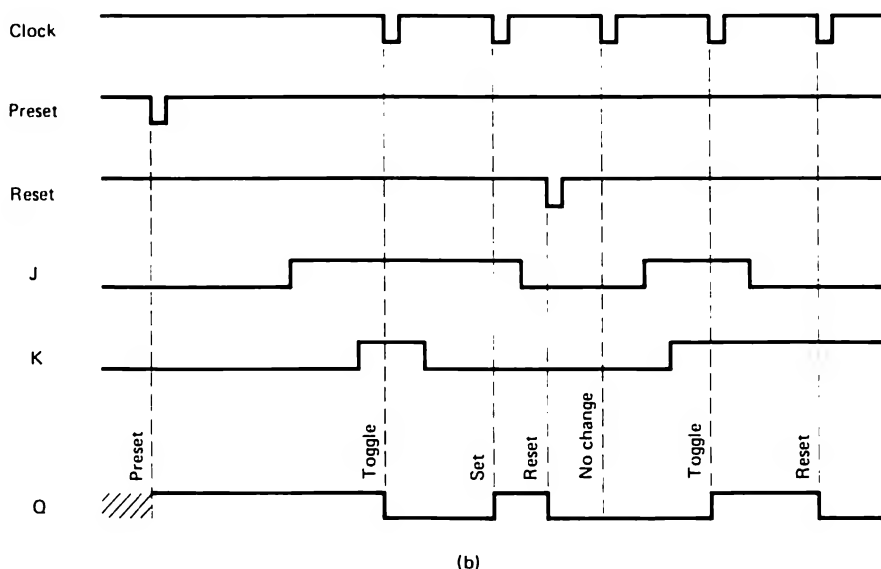
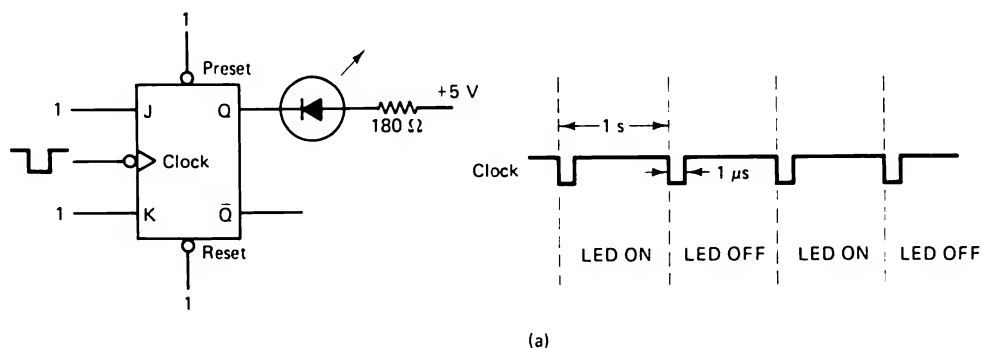


Figure E-2 In (a) the flip-flop is used to detect very short pulses; in (b) it responds to its J and K inputs when the falling edge of the clock pulse occurs.

pulse occurs. Depending on the logic levels of J and K, the Q output may do nothing (no change state), reset ($Q = 0$), set ($Q = 1$), or toggle (switch to the opposite state).

In this book the JK flip-flop is often used to catch pulses that are too short to be observed normally (1 to 2 μs long). Figure E-2a illustrates the technique. With J and K high, the flip-flop will toggle each time the pulse arrives. When Q is low, the LED will light. In this way we can observe each time a pulse occurs even though the pulse itself is only 1 μs wide.

Finally, Fig. E-2b illustrates the general case where all the inputs are changing in time and the flip-flop responds accordingly.

APPENDIX F

IF THE EXPERIMENT DOESN'T WORK

One of *Murphy's laws* states: If anything can go wrong, it will. As you do the experiments in this book you may decide to rewrite this law to say: The more wires you plug into a logic breadboard, the less chance the circuit has of working! This leads us to Rule 1 when working with digital circuits:

Rule 1. Keep calm.

It is surprising how many of yesterday's major problems can be solved the next morning in five minutes with a cool head. When you find yourself getting agitated, take a break. Come back to the circuit later. More often than not, you will fix your problem in a few minutes.

Rule 2. Keep it neat.

This has to be the *cardinal* rule for wiring digital circuits. Don't use a 6-inch length of wire to connect two points that really require only $\frac{1}{2}$ inch of wire. If you don't heed this rule, you'll soon have a "rat's nest" of wires that is impossible to trace and debug. Problems in this kind of circuit almost always require *complete rewiring* of the circuit.

Rule 3. Understand the circuit function.

It is very hard to troubleshoot something if you don't understand how it works in the first place. If you don't know that two 1's into a NAND gate cause a 0 out, you won't be able to recognize a bad chip if you see one. This rule may require you to sit down and study the circuit for awhile. But that's OK. This also helps you to follow Rule 1.

Rule 4. Isolate the problem.

This is the key to all good troubleshooting. Many of the circuits and programs in this book get quite complicated, and when you throw in the computer control function, a number of different problems can occur. A first step is to isolate the problem to *hardware* or *software*. You can do this

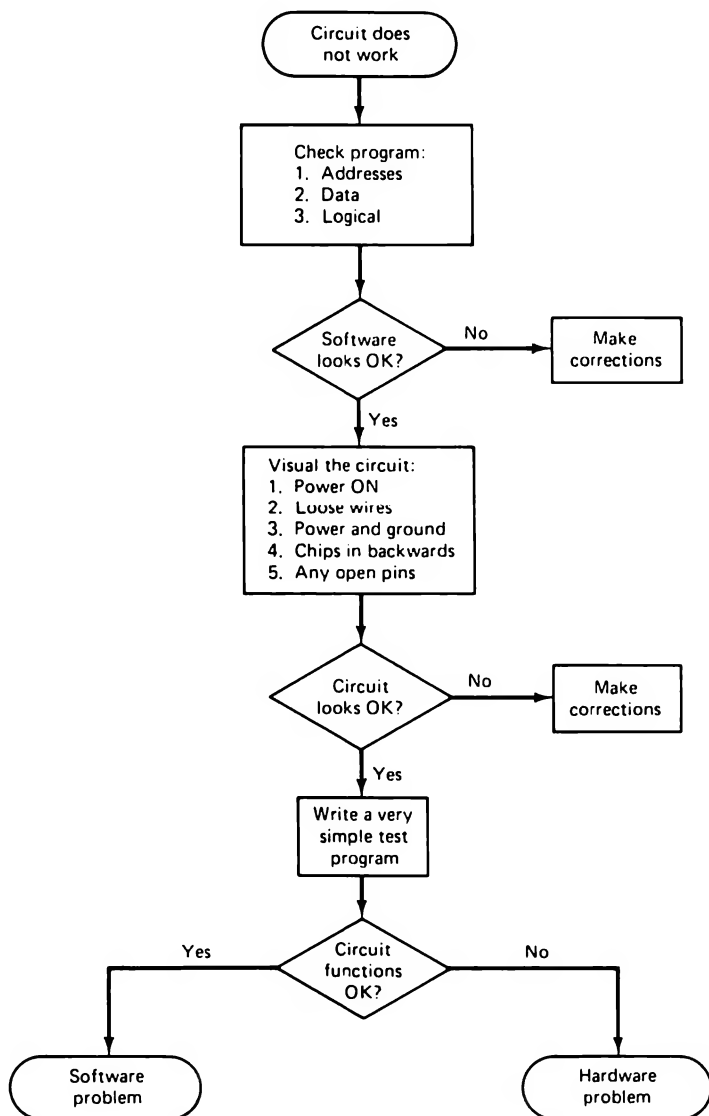


Figure F-1 General flowchart to be used for locating hardware or software problems in the experiments.

by running a *very simple* program that you *know* must work. If the circuit still does not function, it is most likely that you have a *hardware* problem.

Figure F-1 is a flowchart of the logical steps to follow when troubleshooting any of the experiments in this book. After performing obvious checks of the software and hardware for errors, a simple test program should be run. Based on the results of this program, one of two paths is chosen.

SOFTWARE PROBLEMS

If you think the problem is within the software, double check your program line by line for accuracy. Often it is a good idea to insert *breakpoints* into the program and have the computer output *intermediate* results. What you are trying to do is isolate the problem to a particular portion or command in the program.

HARDWARE PROBLEMS

Hardware problems can usually be attributed to faulty wiring or bad chips. A *logic probe* is extremely handy for debugging these types of problems. If you do not have a logic probe, a logic-level tester can be built from a single inverter and LED as shown in Fig. 9-5.

If possible, set up a *static* (nonchanging) condition and trace through the logic with your probe. For example, many of the experiments utilize the 8255 programmable peripheral interface. You could program all I/O pins to be outputs and then see if each pin can be set high and low by testing with your probe. You can similarly test these pins as inputs by applying +5 V or 0 V to each pin (refer to steps 1 and 2 and Question 9-1 of Experiment 9).

All the experiments require an address decoder and connections to the control bus. With your test program running, you should see pulses on the appropriate control lines, and the address decoder output should pulse when its address appears on the bus.

Be very careful when inserting the Vector card into one of the Apple motherboard peripheral connectors. With the card edge connector in place, it is a tight fit. If the computer power will not come on or the computer does not respond to the keyboard, turn the Apple off and then check to make certain that the card is securely in place. The insertion of the Vector card should in no way interfere with normal operation of your Apple.

APPENDIX G

DATA SHEETS



Silicon Gate MOS 8255

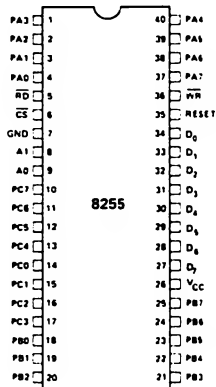
PROGRAMMABLE PERIPHERAL INTERFACE

- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with MCS™-8 and MCS™-80 Microprocessor Families
- Direct Bit Set/Reset Capability
- Easing Control Application Interface
- 40 Pin Dual In-Line Package
- Reduces System Package Count

The 8255 is a general purpose programmable I/O device designed for use with both the 8008 and 8080 microprocessors. It has 24 I/O pins which may be individually programmed in two groups of twelve and used in three major modes of operation. In the first mode (Mode 0), each group of twelve I/O pins may be programmed in sets of 4 to be input or output. In Mode 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining four pins three are used for handshaking and interrupt control signals. The third mode of operation (Mode 2) is a Bidirectional Bus mode which uses 8 lines for a bidirectional bus, and five lines, borrowing one from the other group, for handshaking.

Other features of the 8255 include bit set and reset capability and the ability to source 1mA of current at 1.5 volts. This allows darlington transistors to be directly driven for applications such as printers and high voltage displays.

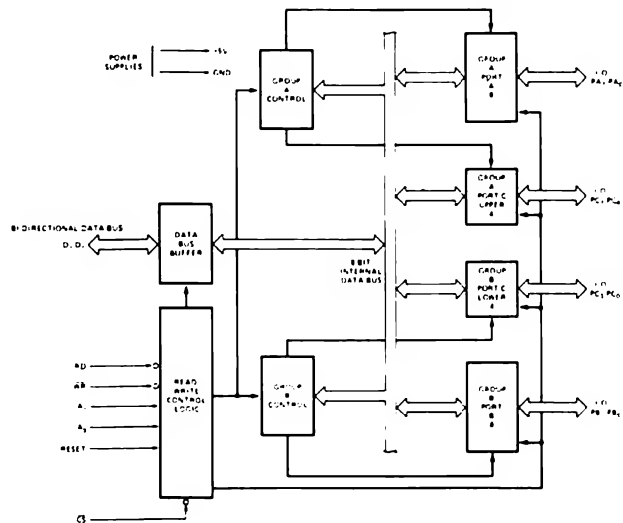
PIN CONFIGURATION



PIN NAMES

D ₇ -D ₀	DATA BUS (BI DIRECTIONAL)
RESET	RESET INPUT
CS	CHIP SELECT
RD	READ INPUT
WR	WRITE INPUT
A ₀ , A ₁	PORT ADDRESS
PA ₇ -PA ₀	PORT A (BIT)
PB ₇ -PB ₀	PORT B (BIT)
PC ₇ -PC ₀	PORT C (BIT)
V _{CC}	+5 VOLTS
GND	0 VOLTS

8255 BLOCK DIAGRAM



SILICON GATE MOS 8255

8255 BASIC FUNCTIONAL DESCRIPTION

General

The 8255 is a Programmable Peripheral Interface (PPI) device designed for use in 8080 Microcomputer Systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the 8080 system bus. The functional configuration of the 8255 is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state, bi-directional, eight bit buffer is used to interface the 8255 to the 8080 system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions by the 8080 CPU. Control Words and Status information are also transferred through the Data Bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the 8080 CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(\overline{CS})

Chip Select: A "low" on this input pin enables the communication between the 8255 and the 8080 CPU.

(\overline{RD})

Read: A "low" on this input pin enables the 8255 to send the Data or Status information to the 8080 CPU on the Data Bus. In essence, it allows the 8080 CPU to "read from" the 8255.

(\overline{WR})

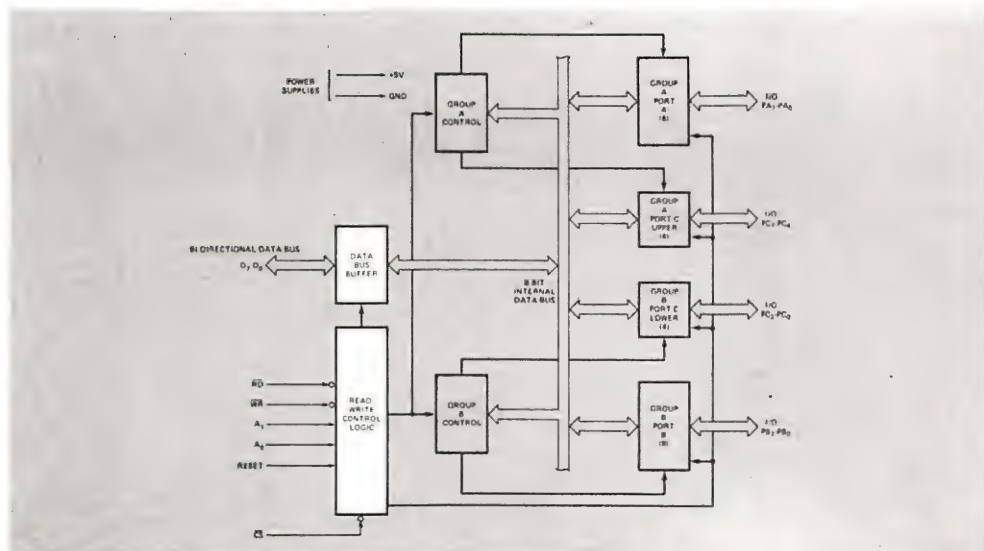
Write: A "low" on this input pin enables the 8080 CPU to write Data or Control words into the 8255.

(A_0 and A_1)

Port Select 0 and Port Select 1: These input signals, in conjunction with the \overline{RD} and \overline{WR} inputs, control the selection of one of the three ports or the Control Word Register. They are normally connected to the least significant bits of the Address Bus (A_0 and A_1).

8255 BASIC OPERATION

A_1	A_0	\overline{RD}	\overline{WR}	\overline{CS}	INPUT OPERATION (READ)
0	0	0	1	0	PORT A \Rightarrow DATA BUS
0	1	0	1	0	PORT B \Rightarrow DATA BUS
1	0	0	1	0	PORT C \Rightarrow DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS \Rightarrow PORT A
0	1	1	0	0	DATA BUS \Rightarrow PORT B
1	0	1	0	0	DATA BUS \Rightarrow PORT C
1	1	1	0	0	DATA BUS \Rightarrow CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS = 3-STATE
1	1	0	1	0	ILLEGAL CONDITION



8255 Block Diagram

SILICON GATE MOS 8255

(RESET)

Reset: A "high" on this input clears all internal registers including the Control Register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the 8080 CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset" etc. that initializes the functional configuration of the 8255.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A — Port A and Port C upper (C7-C4)

Control Group B — Port B and Port C lower (C3-C0)

The Control Word Register can Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

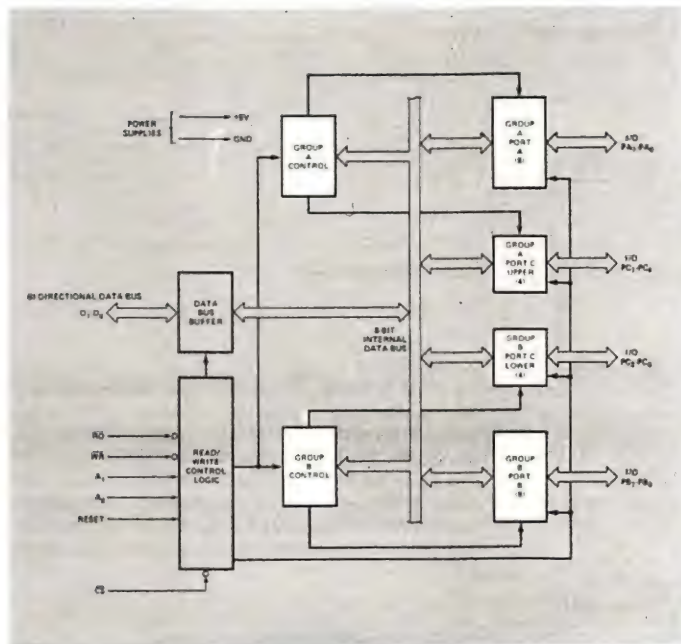
The 8255 contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A: One 8-bit data output latch/buffer and one 8-bit data input latch.

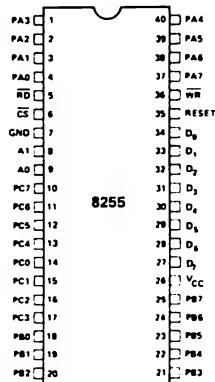
Port B: One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with Ports A and B.

8255 BLOCK DIAGRAM



PIN CONFIGURATION



PIN NAMES

D ₀ -D ₇	DATA BUS (BI DIRECTIONAL)
RESET	RESET INPUT
CS	CHIP SELECT
RD	READ INPUT
WR	WRITE INPUT
A0, A1	PORT ADDRESS
PA7, PA0	PORT A (BIT)
PB7, PB0	PORT B (BIT)
PC7, PC0	PORT C (BIT)
V _{CC}	+5 VOLTS
GND	0 VOLTS

SILICON GATE MOS 8255

8255 DETAILED OPERATIONAL DESCRIPTION

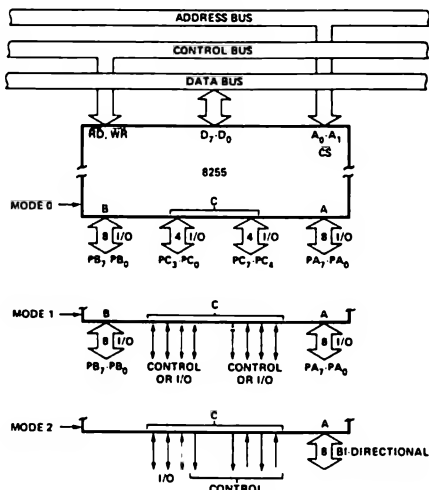
Mode Selection

There are three basic modes of operation that can be selected by the system software:

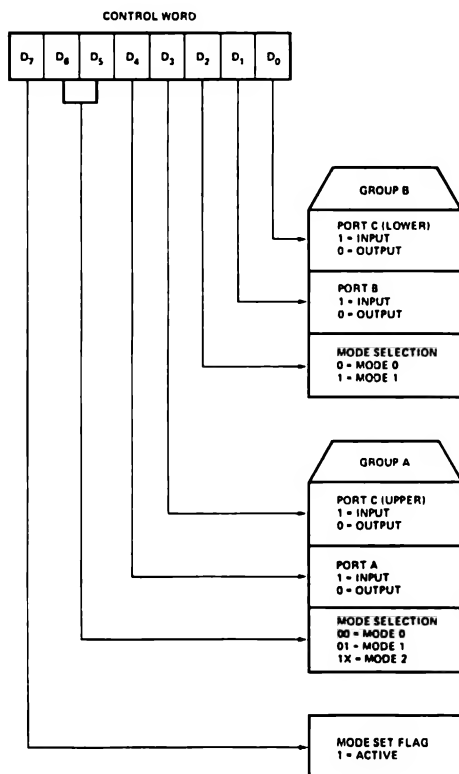
- Mode 0 – Basic Input/Output
- Mode 1 – Strobed Input/Output
- Mode 2 – Bi-Directional Bus

When the RESET input goes "high" all ports will be set to the Input mode (i.e., all 24 lines will be in the high impedance state). After the RESET is removed the 8255 can remain in the Input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single OUTPUT instruction. This allows a single 8255 to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.



Basic Mode Definitions and Bus Interface



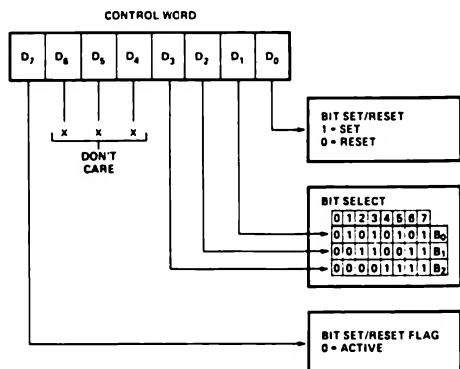
Mode Definition Format

The Mode definitions and possible Mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255 has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTPUT instruction. This feature reduces software requirements in Control-based applications.

SILICON GATE MOS 8255



Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255 is programmed to operate in Mode 1 or Mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from Port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the Bit set/reset function of Port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without effecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) – INTE is SET – Interrupt enable

(BIT-RESET) – INTE is RESET – Interrupt disable

Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

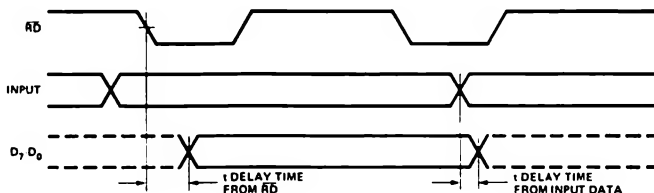
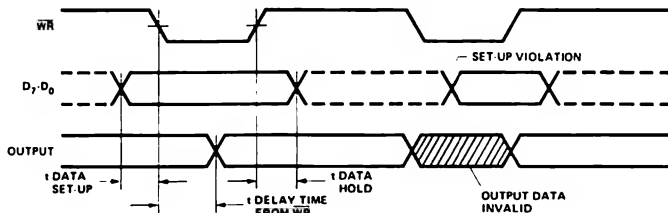
Operating Modes

Mode 0 (Basic Input/Output)

This functional configuration provides simple Input and Output operations for each of the three ports. No "hand-shaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

BASIC INPUT
TIMING (D₇-D₀
FOLLOWS INPUT,
NO LATCHING)BASIC OUTPUT
TIMING (OUTPUTS
LATCHED)

Mode 0 Timing

**MOTOROLA****SEMICONDUCTORS**

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

PERIPHERAL INTERFACE ADAPTER (PIA)

The MC6821 Peripheral Interface Adapter provides the universal means of interfacing peripheral equipment to the M6800 family of microprocessors. This device is capable of interfacing the MPU to peripherals through two 8-bit bidirectional peripheral data buses and four control lines. No external logic is required for interfacing to most peripheral devices.

The functional configuration of the PIA is programmed by the MPU during system initialization. Each of the peripheral data lines can be programmed to act as an input or output, and each of the four control/interrupt lines may be programmed for one of several control modes. This allows a high degree of flexibility in the overall operation of the interface.

- 8-Bit Bidirectional Data Bus for Communication with the MPU
- Two Bidirectional 8-Bit Buses for Interface to Peripherals
- Two Programmable Control Registers
- Two Programmable Data Direction Registers
- Four Individually-Controlled Interrupt Input Lines, Two Usable as Peripheral Control Outputs
- Handshake Control Logic for Input and Output Peripheral Operation
- High-Impedance Three-State and Direct Transistor Drive Peripheral Lines
- Program Controlled Interrupt and Interrupt Disable Capability
- CMOS Drive Capability on Side A Peripheral Lines
- Two TTL Drive Capability on All A and B Side Buffers
- TTL-Compatible
- Static Operation

MAXIMUM RATINGS

Characteristics	Symbol	Value	Unit
Supply Voltage	V_{CC}	0.3 to +7.0	V
Input Voltage	V_{in}	0.3 to +7.0	V
Operating Temperature Range MC6821, MC68A21, MC68B21 MC6821C, MC68A21C, MC68B21C	T_A	T_L to T_H 0 to 70 -40 to +85	°C
Storage Temperature Range	T_{stg}	-55 to +150	°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance			
Ceramic	θ_{JA}	50	°C/W
Plastic		100	
Cerdip		60	

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage (i.e., either V_{SS} or V_{CC}).

MC6821

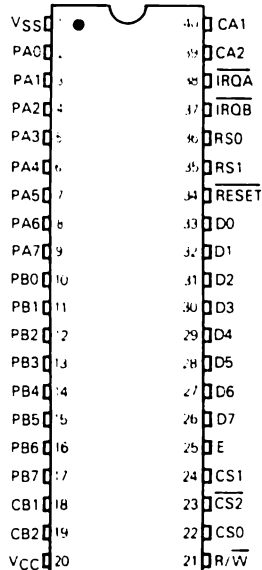
(1.0 MHz)

MC68A21

(1.5 MHz)

MC68B21

(2.0 MHz)

MOS(N-CHANNEL, SILICON-GATE,
DEPLETION LOAD)**PERIPHERAL INTERFACE
ADAPTER****L SUFFIX**
CERAMIC PACKAGE
CASE 715**S SUFFIX**
CERDIP PACKAGE
CASE 734**P SUFFIX**
PLASTIC PACKAGE
CASE 711**PIN ASSIGNMENT**

PIA INTERFACE SIGNALS FOR MPU

The PIA interfaces to the M6800 bus with an 8-bit bidirectional data bus, three chip select lines, two register select lines, two interrupt request lines, a read/write line, an enable line and a reset line. To ensure proper operation with the MC6800, MC6802, or MC6808 microprocessors, VMA should be used as an active part of the address decoding.

Bidirectional Data (D0-D7) — The bidirectional data lines (D0-D7) allow the transfer of data between the MPU and the PIA. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs a PIA read operation. The read/write line is in the read (high) state when the PIA is selected for a read operation.

Enable (E) — The enable pulse, E, is the only timing signal that is supplied to the PIA. Timing of all other signals is referenced to the leading and trailing edges of the E pulse.

Read/Write (R/W) — This signal is generated by the MPU to control the direction of data transfers on the data bus. A low state on the PIA read/write line enables the input buffers and data is transferred from the MPU to the PIA on the E signal if the device has been selected. A high on the read/write line sets up the PIA for a transfer of data to the bus. The PIA output buffers are enabled when the proper address and the enable pulse E are present.

RESET — The active low **RESET** line is used to reset all register bits in the PIA to a logical zero (low). This line can be used as a power-on reset and as a master reset during system operation.

Chip Selects (CS0, CS1, and CS2) — These three input signals are used to select the PIA. CS0 and CS1 must be high and CS2 must be low for selection of the device. Data transfers are then performed under the control of the enable and read/write signals. The chip select lines must be stable

for the duration of the E pulse. The device is deselected when any of the chip selects are in the inactive state.

Register Selects (RS0 and RS1) — The two register select lines are used to select the various registers inside the PIA. These two lines are used in conjunction with internal Control Registers to select a particular register that is to be written or read.

The register and chip select lines should be stable for the duration of the E pulse while in the read or write cycle.

Interrupt Request (IROA and IRQB) — The active low Interrupt Request lines (IROA and IRQB) act to interrupt the MPU either directly or through interrupt priority circuitry. These lines are "open drain" (no load device on the chip). This permits all interrupt request lines to be tied together in a wire-OR configuration.

Each Interrupt Request line has two internal interrupt flag bits that can cause the Interrupt Request line to go low. Each flag bit is associated with a particular peripheral interrupt line. Also, four interrupt enable bits are provided in the PIA which may be used to inhibit a particular interrupt from a peripheral device.

Servicing an interrupt by the MPU may be accomplished by a software routine that, on a prioritized basis, sequentially reads and tests the two control registers in each PIA for interrupt flag bits that are set.

The interrupt flags are cleared (zeroed) as a result of an MPU Read Peripheral Data Operation of the corresponding data register. After being cleared, the interrupt flag bit cannot be enabled to be set until the PIA is deselected during an E pulse. The E pulse is used to condition the interrupt control lines (CA1, CA2, CB1, CB2). When these lines are used as interrupt inputs, at least one E pulse must occur from the inactive edge to the active edge of the interrupt input signal to condition the edge sense network. If the interrupt flag has been enabled and the edge sense circuit has been properly conditioned, the interrupt flag will be set on the next active transition of the interrupt input pin.

PIA PERIPHERAL INTERFACE LINES

The PIA provides two 8-bit bidirectional data buses and four interrupt/control lines for interfacing to peripheral devices.

Section A Peripheral Data (PA0-PA7) — Each of the peripheral data lines can be programmed to act as an input or output. This is accomplished by setting a "1" in the corresponding Data Direction Register bit for those lines which are to be outputs. A "0" in a bit of the Data Direction Register causes the corresponding peripheral data line to act as an input. During an MPU Read Peripheral Data Operation, the data on peripheral lines programmed to act as inputs appears directly on the corresponding MPU Data Bus lines. In the input mode, the internal pullup resistor on these lines represents a maximum of 1.5 standard TTL loads.

The data in Output Register A will appear on the data lines that are programmed to be outputs. A logical "1" written into the register will cause a "high" on the corresponding data

line while a "0" results in a "low." Data in Output Register A may be read by an MPU "Read Peripheral Data A" operation when the corresponding lines are programmed as outputs. This data will be read properly if the voltage on the peripheral data lines is greater than 2.0 volts for a logic "1" output and less than 0.8 volt for a logic "0" output. Loading the output lines such that the voltage on these lines does not reach full voltage causes the data transferred into the MPU on a Read operation to differ from that contained in the respective bit of Output Register A.

Section B Peripheral Data (PB0-PB7) — The peripheral data lines in the B Section of the PIA can be programmed to act as either inputs or outputs in a similar manner to PA0-PA7. They have three-state capability, allowing them to enter a high-impedance state when the peripheral data line is used as an input. In addition, data on the peripheral data lines



PB0-PB7 will be read properly from those lines programmed as outputs even if the voltages are below 2.0 volts for a "high" or above 0.8 V for a "low". As outputs, these lines are compatible with standard TTL and may also be used as a source of up to 1 milliampere at 1.5 volts to directly drive the base of a transistor switch.

Interrupt Input (CA1 and CB1) — Peripheral input lines CA1 and CB1 are input only lines that set the interrupt flags of the control registers. The active transition for these signals is also programmed by the two control registers.

Peripheral Control (CA2) — The peripheral control line CA2 can be programmed to act as an interrupt input or as a

peripheral control output. As an output, this line is compatible with standard TTL; as an input the internal pullup resistor on this line represents 1.5 standard TTL loads. The function of this signal line is programmed with Control Register A.

Peripheral Control (CB2) — Peripheral Control line CB2 may also be programmed to act as an interrupt input or peripheral control output. As an input, this line has high input impedance and is compatible with standard TTL. As an output it is compatible with standard TTL and may also be used as a source of up to 1 milliampere at 1.5 volts to directly drive the base of a transistor switch. This line is programmed by Control Register B.

INTERNAL CONTROLS

INITIALIZATION

A RESET has the effect of zeroing all PIA registers. This will set PA0-PA7, PB0-PB7, CA2 and CB2 as inputs, and all interrupts disabled. The PIA must be configured during the restart program which follows the reset.

There are six locations within the PIA accessible to the MPU data bus: two Peripheral Registers, two Data Direction Registers, and two Control Registers. Selection of these locations is controlled by the RS0 and RS1 inputs together with bit 2 in the Control Register, as shown in Table 1.

Details of possible configurations of the Data Direction and Control Register are as follows:

TABLE 1 — INTERNAL ADDRESSING

RS1	RS0	Control Register Bit		Location Selected
		CRA 2	CRB 2	
0	0	1	X	Peripheral Register A
0	0	0	X	Data Direction Register A
0	1	X	X	Control Register A
1	0	X	1	Peripheral Register B
1	0	X	0	Data Direction Register B
1	1	X	X	Control Register B

X Don't Care

PORT A-B HARDWARE CHARACTERISTICS

As shown in Figure 17, the MC6821 has a pair of I/O ports whose characteristics differ greatly. The A side is designed to drive CMOS logic to normal 30% to 70% levels, and incorporates an internal pullup device that remains connected even in the input mode. Because of this, the A side requires more drive current in the input mode than Port B. In contrast, the B side uses a normal three-state NMOS buffer which cannot pullup to CMOS levels without external resistors. The B side can drive extra loads such as Darlington transistors without problem. When the PIA comes out of reset, the A port represents inputs with pullup resistors, whereas the B side (input mode also) will float high or low, depending upon the load connected to it.

Notice the differences between a Port A and Port B read operation when in the output mode. When reading Port A, the actual pin is read, whereas the B side read comes from an output latch, ahead of the actual pin.

CONTROL REGISTERS (CRA and CRB)

The two Control Registers (CRA and CRB) allow the MPU to control the operation of the four peripheral control lines CA1, CA2, CB1, and CB2. In addition they allow the MPU to enable the interrupt lines and monitor the status of the interrupt flags. Bits 0 through 5 of the two registers may be written or read by the MPU when the proper chip select and register select signals are applied. Bits 6 and 7 of the two registers are read only and are modified by external interrupts occurring on control lines CA1, CA2, CB1, or CB2. The format of the control words is shown in Figure 18.

DATA DIRECTION ACCESS CONTROL BIT (CRA-2 and CRB-2)

Bit 2, in each Control Register (CRA and CRB), determines selection of either a Peripheral Output Register or the corresponding Data Direction E Register when the proper register select signals are applied to RS0 and RS1. A "1" in bit 2 allows access of the Peripheral Interface Register, while a "0" causes the Data Direction Register to be addressed.

Interrupt Flags (CRA-6, CRA-7, CRB-6, and CRB-7) — The four interrupt flag bits are set by active transitions of signals on the four Interrupt and Peripheral Control lines when those lines are programmed to be inputs. These bits cannot be set directly from the MPU Data Bus and are reset indirectly by a Read Peripheral Data Operation on the appropriate section.

Control of CA2 and CB2 Peripheral Control Lines (CRA-3, CRA-4, CRA-5, CRB-3, CRB-4, and CRB-5) — Bits 3, 4, and 5 of the two control registers are used to control the CA2 and CB2 Peripheral Control lines. These bits determine if the control lines will be an interrupt input or an output control signal. If bit CRA-5 (CRB-5) is low, CA2 (CB2) is an interrupt input line similar to CA1 (CB1). When CRA-5 (CRB-5) is high, CA2 (CB2) becomes an output signal that may be used to control peripheral data transfers. When in the output mode, CA2 and CB2 have slightly different loading characteristics.



Control of CA1 and CB1 Interrupt Input Lines (CRA-0, CRB-1, CRA-1, and CRB-1) – The two lowest-order bits of the control registers are used to control the interrupt input lines CA1 and CB1. Bits CRA-0 and CRB-0 are used to

enable the MPU interrupt signals \overline{IROA} and \overline{IROB} , respectively. Bits CRA-1 and CRB-1 determine the active transition of the interrupt input signals CA1 and CB1.

Determine Active CA1 (CB1) Transition for Setting Interrupt Flag IRQA(B)1 — (bit 7)

- b1 = 0: IRQA(B)1 set by high-to-low transition on CA1 (CB1)
b1 = 1: IRQA(B)1 set by low-to-high transition on CA1 (CB1).

IRQA(B) 1 Interrupt Flag (bit 7)

Goes high on active transition of CA1 (CB1). Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

FIGURE 18 — CONTROL WORD FORMAT

CA1 (CB1) Interrupt Request Enable/Disable

- b0 = 0: Disables IRQA(B) MPU Interrupt by CA1 (CB1) active transition.¹
b0 = 1: Enable IRQA(B) MPU Interrupt by CA1 (CB1) active transition

¹ IRQA(B) will occur on next (MPU generated) positive transition of b0 if CA1 (CB1) active transition occurred while interrupt was disabled

Control Register	b7	b6	b5	b4	b3	b2	b1	b0
	IRQA(B)1 Flag	IRQA(B)2 Flag	CA2 (CB2) Control			DDR Access	CA1 (CB1) Control	

IRQA(B)2 Interrupt Flag (bit 6)

When CA2 (CB2) is an input, IRQA(B) goes high on active transition CA2 (CB2); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

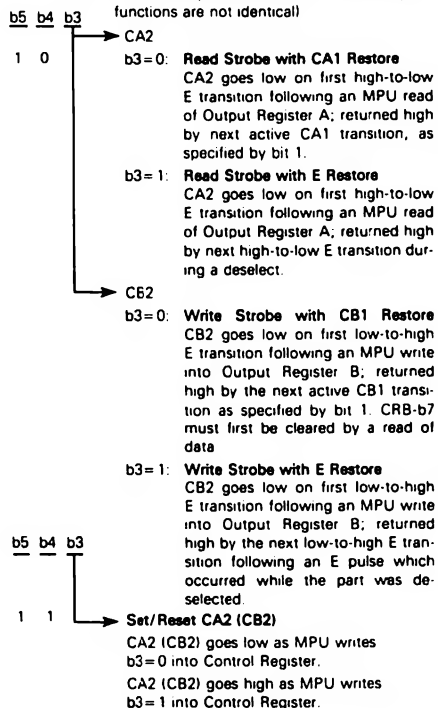
CA2 (CB2) Established as Output (b5 = 1): IRQA(B) 2 = 0, not affected by CA2 (CB2) transitions.

Determines Whether Data Direction Register Or Output Register is Addressed

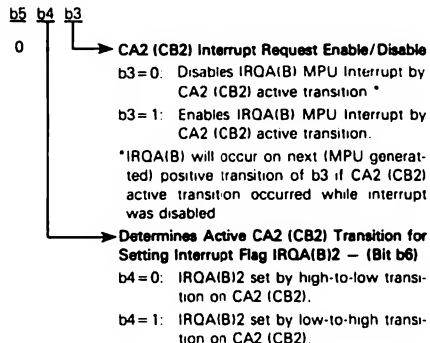
- b2 = 0: Data Direction Register selected.
b2 = 1: Output Register selected.

CA2 (CB2) Established as Output by b5 = 1

(Note that operation of CA2 and CB2 output functions are not identical)



CA2 (CB2) Established as Input by b5 = 0



MC1408, MC1508

GENERAL INFORMATION

Reference Amplifier Drive and Compensation

The reference amplifier provides a voltage at pin 14 for converting the reference voltage to a current, and a turn-around circuit or current mirror for feeding the ladder. The reference amplifier input current, I_{14} , must always flow into pin 14 regardless of the setup method or reference voltage polarity.

Connections for a positive reference voltage are shown in Figure 7. The reference voltage source supplies the full current I_{14} . For bipolar reference signals, as in the multiplying mode, R_{15} can be tied to a negative voltage corresponding to the minimum input level. It is possible to eliminate R_{15} with only a small sacrifice in accuracy and temperature drift. Another method for bipolar inputs is shown in Figure 25.

The compensation capacitor value must be increased with increases in R_{14} to maintain proper phase margin. For R_{14} values of 1.0, 2.5 and 5.0 kilohms, minimum capacitor values are 15, 37, and 75 pF. The capacitor should be tied to V_{EE} as this increases negative supply rejection.

A negative reference voltage may be used if R_{14} is grounded and the reference voltage is applied to R_{15} as shown in Figure 8. A high input impedance is the main advantage of this method. Compensation involves a capacitor to V_{EE} on pin 16, using the values of the previous paragraph. The negative reference voltage must be at least 3.0-volts above the V_{EE} supply. Bipolar input signals may be handled by connecting R_{14} to a positive reference voltage equal to the peak positive input level at pin 15.

When a dc reference voltage is used, capacitive bypass to ground is recommended. The 5.0-V logic supply is not recommended as a reference voltage. If a well regulated 5.0-V supply which drives logic is to be used as the reference, R_{14} should be decoupled by connecting it to +5.0 V through another resistor and bypassing the junction of the two resistors with 0.1 μ F to ground. For reference voltages greater than 5.0 V, a clamp diode is recommended between pin 14 and ground.

If pin 14 is driven by a high impedance such as a transistor current source, none of the above compensation methods apply and the amplifier must be heavily compensated, decreasing the overall bandwidth.

Output Voltage Range

The voltage on pin 4 is restricted to a range of -0.55 to +0.4 volts at +25°C, due to the current switching methods employed in the MC1408. When a current switch is turned "off", the positive voltage on the output terminal can turn "on" the output diode and increase the output current level. When a current switch is turned "on", the negative output voltage range is restricted. The base of the termination circuit Darlington transistor is one diode voltage below ground when pin 1 is grounded, so a negative voltage below the specified safe level will drive the low current device of the Darlington into saturation, decreasing the output current level.

The negative output voltage compliance of the MC1408 may be extended to -5.0 V volts by opening the circuit at pin 1. The negative supply voltage must be more negative than -10 volts. Using a full scale current of 1.992 mA and load resistor of 2.5 kilohms between pin 4 and ground will yield a voltage output of 256 levels between 0 and -4.980 volts. Floating pin 1 does not affect the converter speed or power dissipation. However, the value of the load resistor determines the switching time due to increased voltage swing. Values of R_L up to 500 ohms do not significantly affect performance, but a 2.5-kilohm load increases "worst case" settling time to 1.2 μ s (when all bits are switched on).

Refer to the subsequent text section on Settling Time for more details on output loading.

If a power supply value between -5.0 V and -10 V is desired, a voltage of between 0 and -5.0 V may be applied to pin 1. The value of this voltage will be the maximum allowable negative output swing.

Output Current Range

The output current maximum rating of 4.2 mA may be used only for negative supply voltages typically more negative than -8.0 volts, due to the increased voltage drop across the 350-ohm resistors in the reference current amplifier.

Accuracy

Absolute accuracy is the measure of each output current level with respect to its intended value, and is dependent upon relative accuracy and full scale current drift. Relative accuracy is the measure of each output current level as a fraction of the full scale current. The relative accuracy of the MC1408 is essentially constant with temperature due to the excellent temperature tracking of the monolithic resistor ladder. The reference current may drift with temperature, causing a change in the absolute accuracy of output current. However, the MC1408 has a very low full scale current drift with temperature.

The MC1408/MC1508 Series is guaranteed accurate to within $\pm 1/2$ LSB at +25°C at a full scale output current of 1.992 mA. This corresponds to a reference amplifier output current drive to the ladder network of 2.0 mA, with the loss of one LSB = 8.0 μ A which is the ladder remainder shunted to ground. The input current to pin 14 has a guaranteed value of between 1.9 and 2.1 mA, allowing some mismatch in the NPN current source pair. The accuracy test circuit is shown in Figure 4. The 12-bit converter is calibrated for a full scale output current of 1.992 mA. This is an optional step since the MC1408 accuracy is essentially the same between 1.5 and 2.5 mA. Then the MC1408 circuits' full scale current is trimmed to the same value with R_{14} so that a zero value appears at the error amplifier output. The counter is activated and the error band may be displayed on an oscilloscope, detected by comparators, or stored in a peak detector.

Two 8-bit D-to-A converters may not be used to construct a 16-bit accurate D-to-A converter. 16-bit accuracy implies a total error of $\pm 1/2$ of one part in 65,536, or $\pm 0.00076\%$, which is much more accurate than the $\pm 0.19\%$ specification provided by the MC1408x8.

Multiplying Accuracy

The MC1408 may be used in the multiplying mode with eight-bit accuracy when the reference current is varied over a range of 256:1. The major source of error is the bias current of the termination amplifier. Under "worst case" conditions, these eight amplifiers can contribute a total of 1.6 μ A extra current at the output terminal. If the reference current in the multiplying mode ranges from 16 μ A to 4.0 mA, the 1.6 μ A contributes an error of 0.1 LSB. This is well within eight-bit accuracy referenced to 4.0 mA.

A monotonic converter is one which supplies an increase in current for each increment in the binary word. Typically, the MC1408 is monotonic for all values of reference current above 0.5 mA. The recommended range for operation with a dc reference current is 0.5 to 4.0 mA.



MC1408, MC1508

GENERAL INFORMATION (Continued)

Settling Time

The "worst case" switching condition occurs when all bits are switched "on", which corresponds to a low-to-high transition for all bits. This time is typically 300 ns for settling to within $\pm 1/2$ LSB, for 8-bit accuracy, and 200 ns to $1/2$ LSB for 7 and 6-bit accuracy. The turn off is typically under 100 ns. These times apply when $R_L \leq 500$ ohms and $C_O \leq 25$ pF.

The slowest single switch is the least significant bit, which turns "on" and settles in 250 ns and turns "off" in 80 ns. In applications where the D-to-A converter functions in a positive-going ramp mode, the "worst case" switching condition does not occur, and a settling time of less than 300 ns may be realized. Bit A7 turns "on" in 200 ns and "off" in 80 ns, while bit A6 turns "on" in 150 ns and "off" in 80 ns.

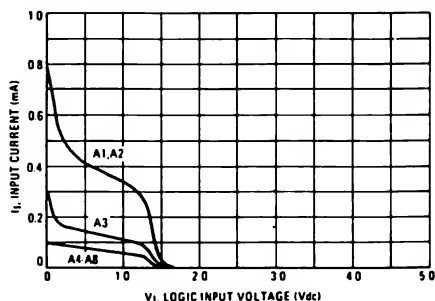
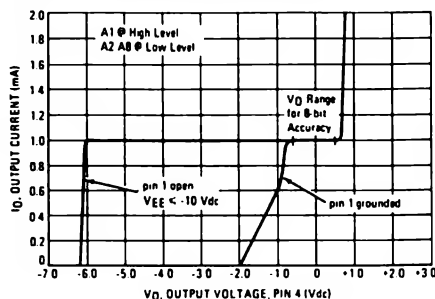
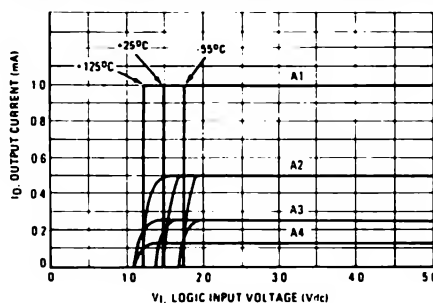
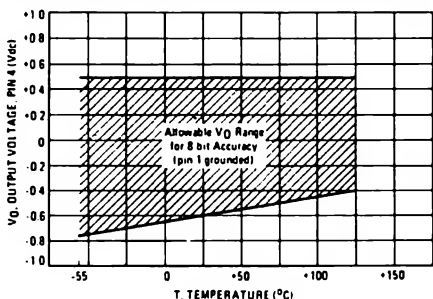
The test circuit of Figure 5 requires a smaller voltage swing for the current switches due to internal voltage clamping in the MC1408. A 1.0-kilohm load resistor from pin 4 to ground gives a typical settling time of 400 ns. Thus, it is voltage swing and not the output RC time constant that determines settling time for most applications.

Extra care must be taken in board layout since this is usually the dominant factor in satisfactory test results when measuring settling time. Short leads, 100 μ F supply bypassing for low frequencies, and minimum scope lead length are all mandatory.

TYPICAL CHARACTERISTICS

($V_{CC} = +5.0$ V, $V_{EE} = -15$ V, $T_A = +25^\circ\text{C}$ unless otherwise noted.)

FIGURE 10 - LOGIC INPUT CURRENT versus INPUT VOLTAGE

FIGURE 12 - OUTPUT CURRENT versus OUTPUT VOLTAGE
(See text for pin 1 restrictions)FIGURE 11 - TRANSFER CHARACTERISTIC versus TEMPERATURE
(A5 thru A8 thresholds lie within range for A1 thru A4)FIGURE 13 - OUTPUT VOLTAGE versus TEMPERATURE
(Negative range with pin 1 open is -5.0 Vdc over full temperature range)



A to D, D to A

ADC0808, ADC0809 8-Bit μ P Compatible A/D Converters With 8-Channel Multiplexer

General Description

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8 single-ended analog signals.

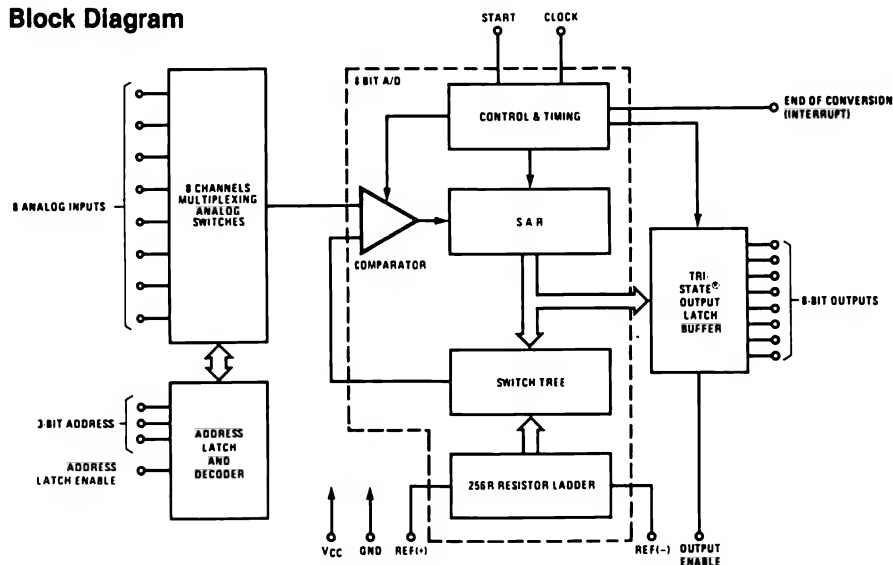
The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE[®] outputs.

The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications. For 16-channel multiplexer with common output (sample/hold port) see ADC0816 data sheet. (See AN-247 for more information.)

Features

- Resolution — 8-bits
- Total unadjusted error — $\pm 1/2$ LSB and ± 1 LSB
- No missing codes
- Conversion time — 100 μ s
- Single supply — 5 V_{DC}
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- 8-channel multiplexer with latched control logic
- Easy interface to all microprocessors, or operates "stand alone"
- Outputs meet T²L voltage level specifications
- 0V to 5V analog input voltage range with single 5V supply
- No zero or full-scale adjust required
- Standard hermetic or molded 28-pin DIP package
- Temperature range — 40°C to +85°C or —55°C to +125°C
- Low power consumption — 15 mW
- Latched TRI-STATE[®] output

Block Diagram



TRI-STATE[®] is a registered trademark of National Semiconductor Corp.

Functional Description

Multiplexer: The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. Table I shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

TABLE I

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

CONVERTER CHARACTERISTICS

The Converter

The heart of this single chip data acquisition system is its 8-bit analog-to-digital converter. The converter is designed

to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach (Figure 1) was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256R network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in Figure 1 are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached $+1/2$ LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n -iterations are required for an n -bit converter. Figure 2 shows a typical example of a 3-bit converter. In the ADC0808, ADC0809, the approximation technique is extended to 8 bits using the 256R network.

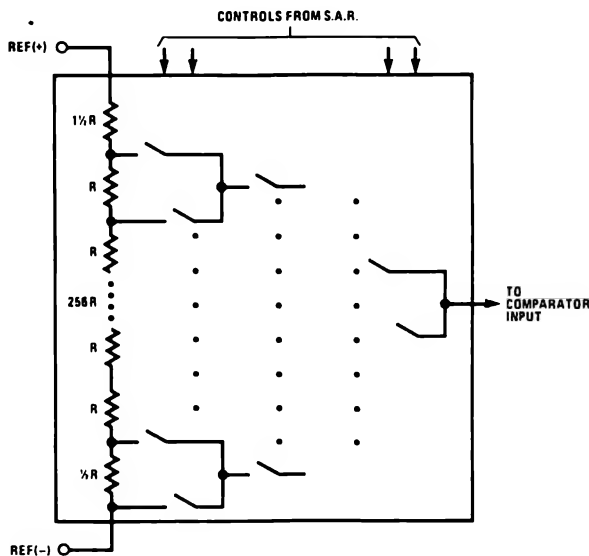


FIGURE 1. Resistor Ladder and Switch Tree

Functional Description (Continued)

The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion.

The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the

comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilized comparator provides the most effective method of satisfying all the converter requirements.

The chopper-stabilized comparator converts the DC input signal into an AC signal. This signal is then fed through a high gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier since the drift is a DC component which is not passed by the AC amplifier. This makes the entire A/D converter extremely insensitive to temperature, long term drift and input offset errors.

Figure 4 shows a typical error curve for the ADC0808 as measured using the procedures outlined in AN-179.

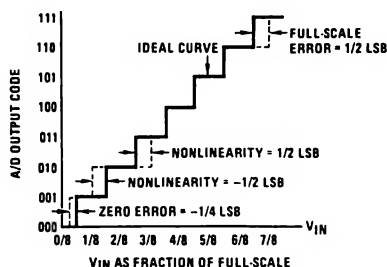


FIGURE 2. 3-Bit A/D Transfer Curve

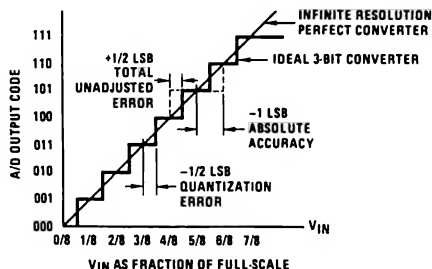


FIGURE 3. 3-Bit A/D Absolute Accuracy Curve

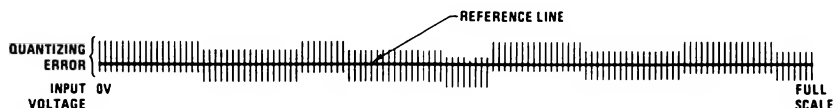
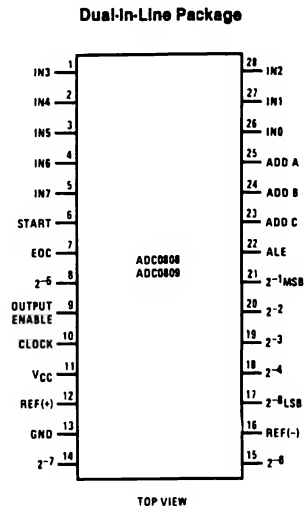


FIGURE 4. Typical Error Curve

Connection Diagram



Timing Diagram

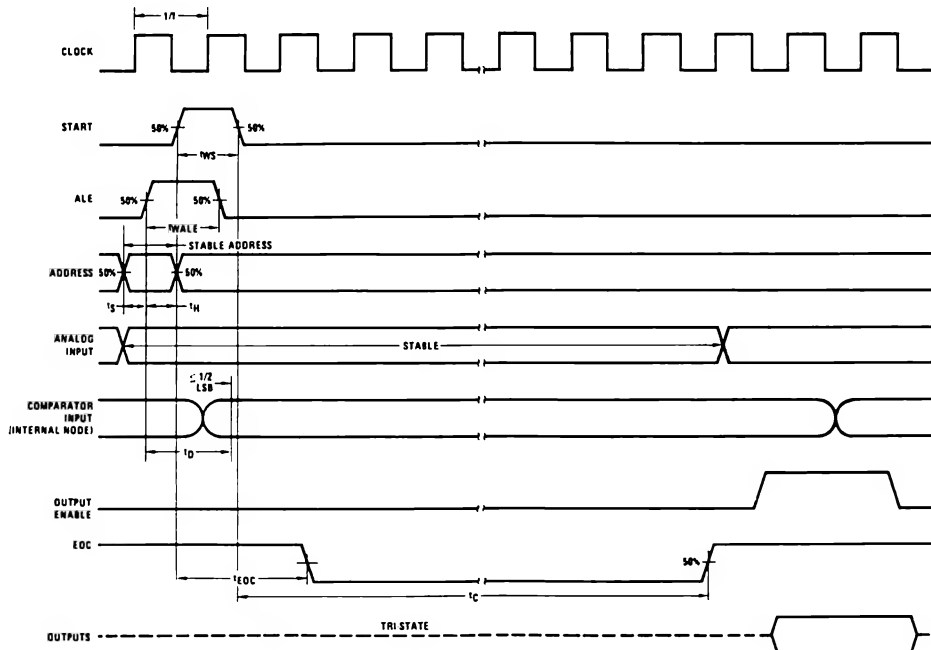


FIGURE 5

**MOTOROLA**

SEMICONDUCTORS

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

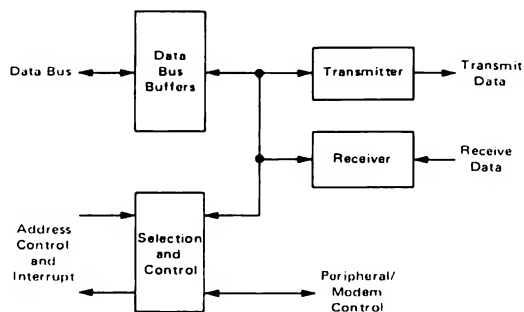
ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (ACIA)

The MC6850 Asynchronous Communications Interface Adapter provides the data formatting and control to interface serial asynchronous data communications information to bus organized systems such as the MC6800 Microprocessing Unit.

The bus interface of the MC6850 includes select, enable, read/write, interrupt and bus interface logic to allow data transfer over an 8-bit bidirectional data bus. The parallel data of the bus system is serially transmitted and received by the asynchronous data interface, with proper formatting and error checking. The functional configuration of the ACIA is programmed via the data bus during system initialization. A programmable Control Register provides variable word lengths, clock division ratios, transmit control, receive control, and interrupt control. For peripheral or modem operation, three control lines are provided. These lines allow the ACIA to interface directly with the MC6860L 0-600 bps digital modem.

- 8- and 9-Bit Transmission
- Optional Even and Odd Parity
- Parity, Overrun and Framing Error Checking
- Programmable Control Register
- Optional +1, +16, and +64 Clock Modes
- Up to 1.0 Mbps Transmission
- False Start Bit Deletion
- Peripheral/Modem Control Functions
- Double Buffered
- One- or Two-Stop Bit Operation

MC6850 ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER BLOCK DIAGRAM

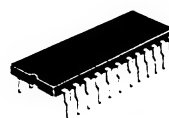


MC6850
(1.0 MHz)
MC68A50
(1.5 MHz)
MC68B50
(2.0 MHz)

MOS

(N-CHANNEL, SILICON-GATE)

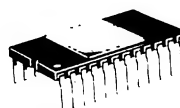
ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER



S SUFFIX
CERDIP PACKAGE
CASE 623



P SUFFIX
PLASTIC PACKAGE
CASE 709



L SUFFIX
CERAMIC PACKAGE
CASE 716

PIN ASSIGNMENT

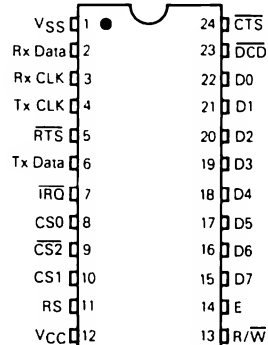
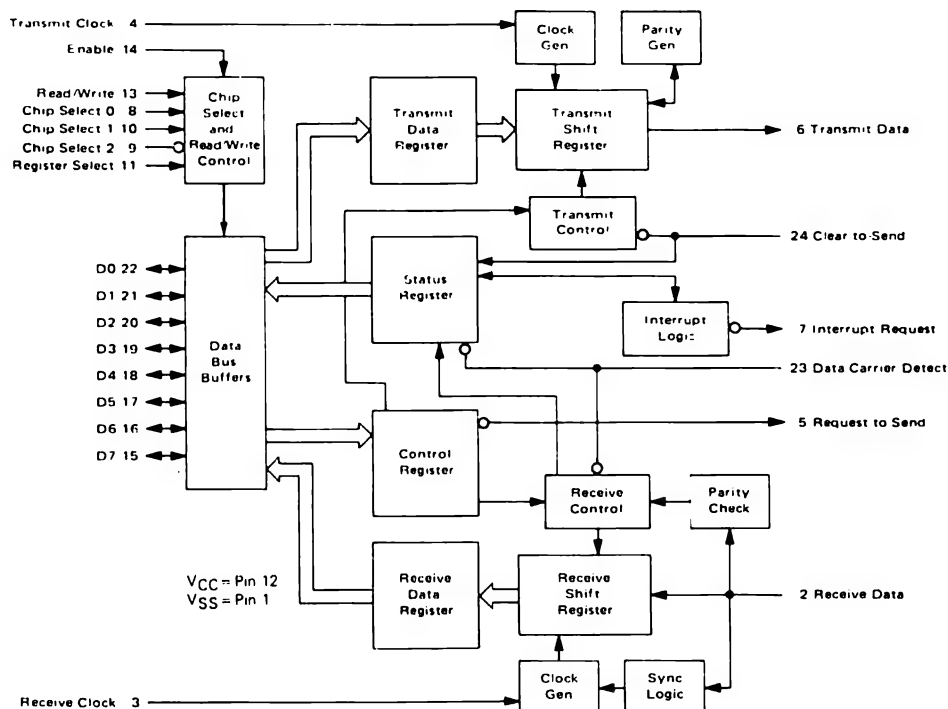


FIGURE 9 — EXPANDED BLOCK DIAGRAM



DEVICE OPERATION

At the bus interface, the ACIA appears as two addressable memory locations. Internally, there are four registers: two read-only and two write-only registers. The read-only registers are Status and Receive Data; the write-only registers are Control and Transmit Data. The serial interface consists of serial input and output lines with independent clocks, and three peripheral/modem control lines.

POWER ON/MASTER RESET

The master reset (CR0, CR1) should be set during system initialization to insure the reset condition and prepare for programming the ACIA functional configuration when the communications channel is required. During the first master reset, the IRQ and RTS outputs are held at level 1. On all other master resets, the RTS output can be programmed high or low with the IRQ output held high. Control bits CR5 and CR6 should also be programmed to define the state of RTS whenever master reset is utilized. The ACIA also contains internal power-on reset logic to detect the power line turn-on transition and hold the chip in a reset state to prevent erroneous output transitions prior to initialization. This circuitry depends on clean power turn-on transitions. The

power-on reset is released by means of the bus-programmed master reset which must be applied prior to operating the ACIA. After master resetting the ACIA, the programmable Control Register can be set for a number of options such as variable clock divider ratios, variable word length, one or two stop bits, parity (even, odd, or none), etc.

TRANSMIT

A typical transmitting sequence consists of reading the ACIA Status Register either as a result of an interrupt or in the ACIA's turn in a polling sequence. A character may be written into the Transmit Data Register if the status read operation has indicated that the Transmit Data Register is empty. This character is transferred to a Shift Register where it is serialized and transmitted from the Transmit Data output preceded by a start bit and followed by one or two stop bits. Internal parity (odd or even) can be optionally added to the character and will occur between the last data bit and the first stop bit. After the first character is written in the Data Register, the Status Register can be read again to check for a Transmit Data Register Empty condition and current peripheral status. If the register is empty, another character can be loaded for transmission even though the first character is in the process of being transmitted (because of



double buffering). The second character will be automatically transferred into the Shift Register when the first character transmission is completed. This sequence continues until all the characters have been transmitted.

RECEIVE

Data is received from a peripheral by means of the Receive Data input. A divide-by-one clock ratio is provided for an externally synchronized clock (to its data) while the divide-by-16 and 64 ratios are provided for internal synchronization. Bit synchronization in the divide-by-16 and 64 modes is initiated by the detection of 8 or 32 low samples on the receive line in the divide-by-16 and 64 modes respectively. False start bit deletion capability insures that a full half bit of a start bit has been received before the internal clock is synchronized to the bit time. As a character is being received, parity (odd or even) will be checked and the error indication will be available in the Status Register along with framing error, overrun error, and Receive Data Register full. In a typical receiving sequence, the Status Register is read to determine if a character has been received from a peripheral. If the Receiver Data Register is full, the character is placed on the 8-bit ACIA bus when a Read Data command is received from the MPU. When parity has been selected for a 7-bit word (7 bits plus parity), the receiver strips the parity bit ($D7 = 0$) so that data alone is transferred to the MPU. This feature reduces MPU programming. The Status Register can continue to be read to determine when another character is available in the Receive Data Register. The receiver is also double buffered so that a character can be read from the data register as another character is being received in the shift register. The above sequence continues until all characters have been received.

INPUT/OUTPUT FUNCTIONS

ACIA INTERFACE SIGNALS FOR MPU

The ACIA interfaces to the M6800 MPU with an 8-bit bidirectional data bus, three chip select lines, a register select line, an interrupt request line, read/write line, and enable line. These signals permit the MPU to have complete control over the ACIA.

ACIA Bidirectional Data ($D0-D7$) — The bidirectional data lines ($D0-D7$) allow for data transfer between the ACIA and the MPU. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs an ACIA read operation.

ACIA Enable (E) — The Enable signal, E , is a high-impedance TTL-compatible input that enables the bus input/output data buffers and clocks data to and from the ACIA. This signal will normally be a derivative of the MC6800 $\phi 2$ Clock or MC6809 E clock.

Read/Write (R/\bar{W}) — The Read/Write line is a high-impedance input that is TTL compatible and is used to control the direction of data flow through the ACIA's input/output data bus interface. When Read/Write is high (MPU Read cycle), ACIA output drivers are turned on and a selected register is read. When it is low, the ACIA output drivers are

turned off and the MPU writes into a selected register. Therefore, the Read/Write signal is used to select read-only or write-only registers within the ACIA.

Chip Select ($CS0, CS1, \bar{CS2}$) — These three high-impedance TTL-compatible input lines are used to address the ACIA. The ACIA is selected when $CS0$ and $CS1$ are high and $\bar{CS2}$ is low. Transfers of data to and from the ACIA are then performed under the control of the Enable Signal, Read/Write, and Register Select.

Register Select (RS) — The Register Select line is a high-impedance input that is TTL compatible. A high level is used to select the Transmit/Receive Data Registers and a low level the Control/Status Registers. The Read/Write signal line is used in conjunction with Register Select to select the read-only or write-only register in each register pair.

Interrupt Request (\bar{IRQ}) — Interrupt Request is a TTL-compatible, open-drain (no internal pullup), active low output that is used to interrupt the MPU. The \bar{IRQ} output remains low as long as the cause of the interrupt is present and the appropriate interrupt enable within the ACIA is set. The \bar{IRQ} status bit, when high, indicates the \bar{IRQ} output is in the active state.

Interrupts result from conditions in both the transmitter and receiver sections of the ACIA. The transmitter section causes an interrupt when the Transmitter Interrupt Enabled condition is selected ($CR5 \cdot CR6$), and the Transmit Data Register Empty ($TDRE$) status bit is high. The $TDRE$ status bit indicates the current status of the Transmitter Data Register except when inhibited by Clear-to-Send (CTS) being high or the ACIA being maintained in the Reset condition. The interrupt is cleared by writing data into the Transmit Data Register. The interrupt is masked by disabling the Transmitter Interrupt via $CR5$ or $CR6$ or by the loss of CTS which inhibits the $TDRE$ status bit. The Receiver section causes an interrupt when the Receiver Interrupt Enable is set and the Receive Data Register Full ($RDRF$) status bit is high, an Overrun has occurred, or Data Carrier Detect (\bar{DCD}) has gone high. An interrupt resulting from the $RDRF$ status bit can be cleared by reading data or resetting the ACIA. Interrupts caused by Overrun or loss of \bar{DCD} are cleared by reading the status register after the error condition has occurred and then reading the Receive Data Register or resetting the ACIA. The receiver interrupt is masked by resetting the Receiver Interrupt Enable.

CLOCK INPUTS

Separate high-impedance TTL-compatible inputs are provided for clocking of transmitted and received data. Clock frequencies of 1, 16, or 64 times the data rate may be selected.

Transmit Clock ($Tx CLK$) — The Transmit Clock input is used for the clocking of transmitted data. The transmitter initiates data on the negative transition of the clock.

Receive Clock ($Rx CLK$) — The Receive Clock input is used for synchronization of received data. (In the ~ 1 mode, the clock and data must be synchronized externally.) The receiver samples the data on the positive transition of the clock.



SERIAL INPUT/OUTPUT LINES

Receive Data (Rx Data) — The Receive Data line is a high-impedance TTL-compatible input through which data is received in a serial format. Synchronization with a clock for detection of data is accomplished internally when clock rates of 16 or 64 times the bit rate are used.

Transmit Data (Tx Data) — The Transmit Data output line transfers serial data to a modem or other peripheral.

PERIPHERAL/MODEM CONTROL

The ACIA includes several functions that permit limited control of a peripheral or modem. The functions included are Clear-to-Send, Request-to-Send and Data Carrier Detect.

Clear-to-Send (CTS) — This high-impedance TTL-compatible input provides automatic control of the transmitting end of a communications link via the modem Clear-to-Send active low output by inhibiting the Transmit Data Register Empty (TDRE) status bit.

Request-to-Send (RTS) — The Request-to-Send output enables the MPU to control a peripheral or modem via the data bus. The RTS output corresponds to the state of the Control Register bits CR5 and CR6. When CR6=0 or both CR5 and CR6=1, the RTS output is low (the active state). This output can also be used for Data Terminal Ready (DTR).

Data Carrier Detect (DCD) — This high-impedance TTL-compatible input provides automatic control, such as in the receiving end of a communications link by means of a modem Data Carrier Detect output. The DCD input inhibits and initializes the receiver section of the ACIA when high. A low-to-high transition of the Data Carrier Detect initiates an interrupt to the MPU to indicate the occurrence of a loss of carrier when the Receive Interrupt Enable bit is set. The Rx CLK must be running for proper DCD operation.

ACIA REGISTERS

The expanded block diagram for the ACIA indicates the internal registers on the chip that are used for the status, control, receiving, and transmitting of data. The content of each of the registers is summarized in Table 1.

TRANSMIT DATA REGISTER (TDR)

Data is written in the Transmit Data Register during the negative transition of the enable (E) when the ACIA has been addressed with RS high and R/W low. Writing data into the register causes the Transmit Data Register Empty bit in the Status Register to go low. Data can then be transmitted. If the transmitter is idling and no character is being transmitted, then the transfer will take place within 1-bit time of the trailing edge of the Write command. If a character is being transmitted, the new data character will commence as soon as the previous character is complete. The transfer of data causes the Transmit Data Register Empty (TDRE) bit to indicate empty.

RECEIVE DATA REGISTER (RDR)

Data is automatically transferred to the empty Receive Data Register (RDR) from the receiver deserializer (a shift register) upon receiving a complete character. This event causes the Receive Data Register Full bit (RDRF) in the status buffer to go high (full). Data may then be read through the bus by addressing the ACIA and selecting the Receive Data Register with RS and R/W high when the ACIA is enabled. The non-destructive read cycle causes the RDRF bit to be cleared to empty although the data is retained in the RDR. The status is maintained by RDRF as to whether or not the data is current. When the Receive Data Register is full, the automatic transfer of data from the Receiver Shift Register to the Data Register is inhibited and the RDR contents remain valid with its current status stored in the Status Register.

TABLE 1 — DEFINITION OF ACIA REGISTER CONTENTS

Data Bus Line Number	Buffer Address			
	RS • R/W	RS • R/W	RS • R/W	RS • R/W
	Transmit Data Register	Receive Data Register	Control Register	Status Register
	(Write Only)	(Read Only)	(Write Only)	(Read Only)
0	Data Bit 0*	Data Bit 0	Counter Divide Select 1 (CR0)	Receive Data Register Full (RDRF)
1	Data Bit 1	Data Bit 1	Counter Divide Select 2 (CR1)	Transmit Data Register Empty (TDRE)
2	Data Bit 2	Data Bit 2	Word Select 1 (CR2)	Data Carrier Detect (DCD)
3	Data Bit 3	Data Bit 3	Word Select 2 (CR3)	Clear to Send (CTS)
4	Data Bit 4	Data Bit 4	Word Select 3 (CR4)	Framing Error (FE)
5	Data Bit 5	Data Bit 5	Transmit Control 1 (CR5)	Receiver Overrun (OVRN)
6	Data Bit 6	Data Bit 6	Transmit Control 2 (CR6)	Parity Error (PE)
7	Data Bit 7**	Data Bit 7**	Receive Interrupt Enable (CR7)	Interrupt Request (IRQ)

* Leading bit = LSB Bit 0

** Data bit will be zero in 7 bit plus parity modes

*** Data bit is "don't care" in 7 bit plus parity modes



MOTOROLA Semiconductor Products Inc.

CONTROL REGISTER

The ACIA Control Register consists of eight bits of write-only buffer that are selected when RS and R/W are low. This register controls the function of the receiver, transmitter, interrupt enables, and the Request-to-Send peripheral/modem control output.

Counter Divide Select Bits (CR0 and CR1) — The Counter Divide Select Bits (CR0 and CR1) determine the divide ratios utilized in both the transmitter and receiver sections of the ACIA. Additionally, these bits are used to provide a master reset for the ACIA which clears the Status Register (except for external conditions on CTS and DCD) and initializes both the receiver and transmitter. Master reset does not affect other Control Register bits. Note that after power-on or a power fail/restart, these bits must be set high to reset the ACIA. After resetting, the clock divide ratio may be selected. These counter select bits provide for the following clock divide ratios:

CR1	CR0	Function
0	0	÷ 1
0	1	÷ 16
1	0	÷ 64
1	1	Master Reset

Word Select Bits (CR2, CR3, and CR4) — The Word Select bits are used to select word length, parity, and the number of stop bits. The encoding format is as follows:

CR4	CR3	CR2	Function
0	0	0	7 Bits + Even Parity + 2 Stop Bits
0	0	1	7 Bits + Odd Parity + 2 Stop Bits
0	1	0	7 Bits + Even Parity + 1 Stop Bit
0	1	1	7 Bits + Odd Parity + 1 Stop Bit
1	0	0	8 Bits + 2 Stop Bits
1	0	1	8 Bits + 1 Stop Bit
1	1	0	8 Bits + Even parity + 1 Stop Bit
1	1	1	8 Bits + Odd Parity + 1 Stop Bit

Word length, Parity Select, and Stop Bit changes are not buffered and therefore become effective immediately.

Transmitter Control Bits (CR5 and CR6) — Two Transmitter Control bits provide for the control of the interrupt from the Transmit Data Register Empty condition, the Request-to-Send (RTS) output, and the transmission of a Break level (space). The following encoding format is used:

CR6	CR5	Function
0	0	RTS = low, Transmitting Interrupt Disabled.
0	1	RTS = low, Transmitting Interrupt Enabled.
1	0	RTS = high, Transmitting Interrupt Disabled.
1	1	RTS = low, Transmits a Break level on the Transmit Data Output. Transmitting Interrupt Disabled.

Receive Interrupt Enable Bit (CR7) — The following interrupts will be enabled by a high level in bit position 7 of the Control Register (CR7): Receive Data Register Full, Overrun, or a low-to-high transition on the Data Carrier Detect (DCD) signal line.

STATUS REGISTER

Information on the status of the ACIA is available to the MPU by reading the ACIA Status Register. This read-only register is selected when RS is low and R/W is high. Information stored in this register indicates the status of the Transmit Data Register, the Receive Data Register and error logic, and the peripheral/modem status inputs of the ACIA.

Receive Data Register Full (RDRF), Bit 0 — Receive Data Register Full indicates that received data has been transferred to the Receive Data Register. RDRF is cleared after an MPU read of the Receive Data Register or by a master reset. The cleared or empty state indicates that the contents of the Receive Data Register are not current. Data Carrier Detect being high also causes RDRF to indicate empty.

Transmit Data Register Empty (TDRE), Bit 1 — The Transmit Data Register Empty bit being set high indicates that the Transmit Data Register contents have been transferred and that new data may be entered. The low state indicates that the register is full and that transmission of a new character has not begun since the last write data command.

Data Carrier Detect (DCD), Bit 2 — The Data Carrier Detect bit will be high when the DCD input from a modem has gone high to indicate that a carrier is not present. This bit going high causes an Interrupt Request to be generated when the Receive Interrupt Enable is set. It remains high after the DCD input is returned low until cleared by first reading the Status Register and then the Data Register or until a master reset occurs. If the DCD input remains high after read status and read data or master reset has occurred, the interrupt is cleared, the DCD status bit remains high and will follow the DCD input.

Clear-to-Send (CTS), Bit 3 — The Clear-to-Send bit indicates the state of the Clear-to-Send input from a modem. A low CTS indicates that there is a Clear-to-Send from the modem. In the high state, the Transmit Data Register Empty bit is inhibited and the Clear-to-Send status bit will be high. Master reset does not affect the Clear-to-Send status bit.

Framing Error (FE), Bit 4 — Framing error indicates that the received character is improperly framed by a start and a stop bit and is detected by the absence of the first stop bit. This error indicates a synchronization error, faulty transmission, or a break condition. The framing error flag is set or reset during the receive data transfer time. Therefore, this error indicator is present throughout the time that the associated character is available.

Receiver Overrun (OVRN), Bit 5 — Overrun is an error flag that indicates that one or more characters in the data stream were lost. That is, a character or a number of characters were received but not read from the Receive Data Register (RDR) prior to subsequent characters being received. The overrun condition begins at the midpoint of the last bit of the second character received in succession without a read of the RDR having occurred. The Overrun does not occur in the Status Register until the valid character prior to Overrun has



MC6850•MC68A50•MC68B50

been read. The RDRF bit remains set until the Overrun is reset. Character synchronization is maintained during the Overrun condition. The Overrun indication is reset after the reading of data from the Receive Data Register or by a Master Reset.

Parity Error (PE), Bit 6 — The parity error flag indicates that the number of highs (ones) in the character does not agree with the preselected odd or even parity. Odd parity is defined to be when the total number of ones is odd. The parity error indication will be present as long as the data

character is in the RDR. If no parity is selected, then both the transmitter parity generator output and the receiver parity check results are inhibited.

Interrupt Request (\overline{IRQ}), Bit 7 — The \overline{IRQ} bit indicates the state of the \overline{IRQ} output. Any interrupt condition with its applicable enable will be indicated in this status bit. Anytime the \overline{IRQ} output is low the \overline{IRQ} bit will be high to indicate the interrupt or service request status. \overline{IRQ} is cleared by a read operation to the Receive Data Register or a write operation to the Transmit Data Register.

INDEX

A

AC control interface, 105-9 (*see also* Relay interface)
Address bus, 14-15, 23
Address decoder, 26-31
 switch selectable, 43-44
Analog-to-digital converter (ADC):
 description, 127-29
 flash converter, 129-30
 National Semiconductor
 ADC0809, 132-35 (*see also* Appendix G):
 Apple II interface, 133-35
 description, 132-33
 digital voltmeter, 135-36
 temperature sensor, 92-93
 successive approximations converter, 131-32
 tracking converter, 130
AND gate, 202-4
Apple II:
 address decoding, 30
 memory map, 24-25
 slot connectors, 4-5
 using the game connector:
 input, 95-97
 output, 109-11

ASCII, 166-67 (Table 12-2)
Asynchronous Communications
 Interface Adapter (ACIA), 156-69
 Motorola MC6850 (*see also* Appendix G):
 Apple II interface, 159-61
 control register, 161-63
 control software, 164-66
 modem control lines, 169
 status register, 163-64

B

Baud rate, 155-56
Baud rate generator (MC14411), 161-62
Binary number system, 199-201
Breadboard, 6-7 (Table I-1)
Bus:
 address, 14-15, 23
 control, 14-15, 24
 data, 14-15, 23
 three bus architecture, 14-15
Buzzer, piezoelectric, 106

C

Clock, 15-16
 Comparator:
 light sensing interface, 89-92
 LM339, 89-90
 temperature sensing interface,
 92-93
 Connector descriptions, 191-93
 Control bus, 14-15, 24

D

Data bus, 14-15, 23
 Decimal-to-binary conversion
 (BASIC program), 51-52
 Decoder, 26-31
 Device select, 29-31
 Digital communications, 166-69
 Digital-to-analog converter (DAC):
 characteristics, 122-23
 (Table 9-1)
 full-scale output, 116
 hypothetical 4-bit, 115-17
 Motorola MC1408, 117-18 (*see*
 also Appendix G):
 Apple II interface, 117-20
 control software, 119-21
 programmable power supply,
 124-25
 waveform generator, 124

F

Flash converter, 129-30
 Flip-flop, 205-6
 Furnace controller, 107

G

Game connector:
 used for input, 95-97
 used for output, 109-11
 General Instrument:
 AY-3-8910, 178
 AY-5-1013, 156-57
 Graphics, use of high resolution,
 137

H

Handshaking, 58, 70, 145-47

I

Input port, 47-49, 53
 Intel 8255, 57 (*see also*
 Appendix G)
 Interface:
 AC control, 105-9
 ACIA, 159-61
 ADC, 133-35
 DAC, 117-20
 digital thermometer, 136-37
 digital voltmeter, 135-36
 furnace controller, 107
 handshaking test circuit, 148-51
 magnetic switch, 87-89
 mechanical relay, 102-5
 organ, electronic using the
 AY-3-8910, 188-90
 peripheral interface adapter
 (PIA), 71-73
 photoresistor, 89-92
 programmable peripheral inter-
 face (PPI), 59-60
 programmable power supply,
 124-25

- smoke detector, 94-95
- solid-state relay, 105-6
- sound generator (AY-3-8910), 178-80
- temperature sensor, 92-93
- traffic light controller, 62-68, 79-82
- ultrasonic, 106-9
- Interrupt driven I/O, 143-45
- I/O mapped I/O, 18
- I/O port enable pulse, 31-33
- I/O types, 141-45

L

- Latch, 37-38 (Table 3-1)
- Logic gates, 202-4
- Logic probe, 7-9 (Table I-2)

M

- Machine cycles, 16-17, 21
- Magnetic switch interface, 87-89
- Masking, 51
- Memory map (Apple II), 24-25
- Memory-mapped I/O, 18
- Modem, 169
- Motorola:
 - MC1408, 117-18 (*see also* Appendix G)
 - MC6820, 70-71 (*see also* Appendix G)
 - MC6850, 156-69 (*see also* Appendix G)
 - MC14411, 161-62

N

- NAND gate, 202-4
- National Semiconductor ADC0809, 132-33 (*see also* Appendix G)
- NOR gate, 202-4

O

- Operational amplifier (Op-Amp):
 - current-to-voltage converter, 119
 - voltage follower, 119
- Opto-coupler, 105-6
- OR gate, 202-4
- Output port, 37-41, 44

P

- Partial decoding, 27-30
- Parts list, 195-98
- Pass transistor, 121-22
- PEEK, 18, 33-35, 47-48
- Peripheral Interface Adapter,
 - Motorola MC6850 (*see also* Appendix G):
 - Apple II interface, 71-73
 - application as traffic light controller, 78-79
 - block diagram, 71
 - control register format, 74-75
 - description, 70-71
 - software control, 73, 76-77
- Photoresistor interface, 89-92
- POKE, 18, 33-35, 37
- Polled I/O, 141-43
- Printer port, 154-55

Programmable Peripheral Interface,
 Intel 8255 (*see also*
 Appendix G):
 Apple II interface, 59-60
 application as traffic light
 controller, 62-65
 block diagram, 58
 handshaking signals, 145-47
 initialization, 61-62, 145-48
 mode 0, 57
 mode 1, 145-48
 modes of operation, 57-58
 Programmable sound generator
 (*see* Sound generator)

R

RAM, 25
 READY flag, 141
 Relay interface:
 mechanical, 102-5
 solid-state, 105-6
 ROM, 25
 RS-232C, 166-69

S

Serial data transmission, 154-56
 Serial interfacing, 156-69
 Seven-segment display:
 software driver, 150
 two digit output port, 40-43
 Smoke detector interface, 94-95
 Solderless breadboard, 6-7
 (Table I-1)
 Sound generator:
 General Instrument AY-3-8910:
 Apple II interface, 178-80
 application as electronic
 organ, 188-90

control with BASIC, 180-87
 description, 178
 summary of functions, 186
 timing, 180-82
 Mostek MK50240N, 173, 175
 Texas Instruments:
 SN76477, 175-76
 SN76489A, 176-77
 Successive approximations ADC,
 131

T

Temperature, analog sensing,
 136-37
 Temperature sensor interface,
 92-93
 Texas Instruments (*see* Sound
 generator)
 Timing diagrams, memory read and
 write, 16-17
 Tone decoder, 108-9
 Tracking ADC, 130
 Transducer, 106-9
 Transistor relay driver, 103-5
 Tri-state gate, 47-48, 50
 Troubleshooting, 207-9

U

Ultrasonic interface, 106-9
 Universal Asynchronous Receiver-
 Transmitter (UART),
 156-57

V

Voltage, analog sensing, 135-36

